TÍTOL: Localització de text en imatges amb sistemes Android

AUTOR: Joel Sala Ramos

DATA: 15 de Abril de 2012

DIRECTOR: Juan Climent Vilaró

DEPARTAMENT DEL DIRECTOR: Departament d'Enginyeria de Sistemes, Automàtica i Informàtica Industrial - ESAII

TITULACIÓ: Enginyeria Informàtica

CENTRE: Facultat d'Informàtica de Barcelona (FIB)

UNIVERSITAT: Universitat Politècnica de Catalunya (UPC)

BarcelonaTech

Índex

1. Introducció 1
   1.1. Descripció del problema 1
   1.2. Restriccions al problema 1
   1.3. Aplicacions del projecte 2
2. Conceptes previs 4
   2.1. Fonaments de la visió per computador 4
      2.1.1. Histogramació 5
      2.1.2. Binarització 5
      2.1.3. Operacions morfològiques 7
      2.1.4. Etiquetatge 11
      2.1.5. Extracció de característiques 12
      2.1.6. Classificació de regions 13
   2.2. Sistema Android 16
      2.2.1. Historia del sistema 16
      2.2.2. Evolució de les versions 17
      2.2.3. Estructura del sistema operatiu 22
3. Estat de l'art 24
   3.1. Aplicacions actuals 24
   3.2. Algoritmes de segmentació de la imatge 26
      3.2.1. Segmentació usant 'Toggle-mapping' 27
      3.2.2. Stroke width transform 27
      3.2.3. Ultimate Opening 28
      3.2.4. Localised Measures 28
   3.3. Detecció de text en imatges segmentades 29
      3.3.1. Filtrat de regions 29
      3.3.2. Detecció de caràcters 29
   3.4. Reconeixement de text(OCR) 30
4. Solució emprada 32
   4.1. Algoritme de segmentació 32
   4.2. Detecció de text 32
   4.3. Reconeixement de text 33
   4.4. Visió global del sistema 33

5. Implementació 35
5.1. Introducció 35
5.2. Entorn de treball 35
5.3. Fase de desenvolupament del treball 38
5.3.1. Obtenir la imatge 39
5.3.2. Millorar la imatge 40
5.3.3. Toggle-mapping 40
5.3.4. Filtrat de regions 42
5.3.5. Millora de regions 42
5.3.6. Classificador 48
5.3.7. Divisió de la imatge 52
5.3.8. Text reconegut 54
5.4. Desenvolupament de l'aplicació en Android 55
5.4.1. Estructura de l'aplicació 56
5.4.2. Obtenir la imatge 58
5.4.3. Processament de la imatge 59
5.4.4. Execució del OCR 61
5.4.5. Interfície gràfica 62
5.4.6. Ús del text reconegut 67
6. Resultats 70
6.1. Detecció de text 71
6.2. Reconeixement de caràcters (OCR) 74
7. Gestió del projecte 76
7.1. Planificació 76
7.2. Pressupost 79
8. Conclusions 81
8.1. Conclusió del projecte 81
8.2. Treball futur 81
8.3. Possibilitats de mercat 82
8.4. Valoració personal 83
9. Referències 84
10. Annexes 86
10.1. Instal·lació de l'entorn de treball 86
10.2. Manual d'usuari DroidText 96

10.3. Codi font 99
10.4. Base de dades 99

Índex de figures

Figura 1 - Imatge Original 1
Figura 2 - Imatge després del processat 1
Figura 3 - Imatge amb reflexos 2
Figura 4 - Icona Google Translate 3
Figura 5 - Icona veu sobre text i accés universal 3
Figura 6 - Imatge en escala de grisos 4
Figura 7 - Histograma de la imatge 5
Figura 8 - Binarització amb diferents llindars globals 6
Figura 9 - Binarització amb llindar global i per regions 7
Figura 10 - Intersecció d'imatges 8
Figura 11 - Unió d'imatges 8
Figura 12 - Complement d'una imatge 9
Figura 13 - Erosió d'una imatge 9
Figura 14 - Dilatació d'una imatge 10
Figura 15 - Opening d'una imatge 10
Figura 16 - Closing d'una imatge 11
Figura 17 - Connectivitat 4 i 8 d'un píxel 12
Figura 18 - Imatge etiquetada 12
Figura 19 - Bounding Box d'un cc 13
Figura 20 - Arquitectura del sistema Android 22
Figura 21 - Prizmo iPhone App 25
Figura 22 - Word Lens iPhone App 25
Figura 23 - Goggles iPhone App 26
Figura 24 - Visió global del sistema 33
Figura 25 - Versions d'Android segons el seu ús 37
Figura 26 - Visió global del algoritme 39
Figura 27 - Imatge obtinguda 39
Figura 28 - Imatge millorada 40
Figura 29 - Imatge binaritzada 41
Figura 30 - Imatge abans i després del filtrat 42
Figura 31 - Regió amb contorn obert 43
Figura 32 - Imatge amb forats omplerts 43
Figura 33 - Etapes de la millora I 44
Figura 34 - Màscares de píxels interiors i exteriors a la regió 45
Figura 35 - Imatge original, píxels exteriors i píxels interiors (de l'original) I 45
Figura 36 - Imatge original, píxels exteriors i píxels interiors (de l'original) II 46
Figura 37 - Etapes de la millora II 46
Figura 38 - Caràcters originals binaritzats 47
Figura 39 - Millora de les regions 47
Figura 40 - Imatges d'entrenament 49
Figura 41 - Arbre de decisió 51
Figura 42 - Imatge filtrada pel classificador 52
Figura 43 - Imatge dividida i pintada segons components connexos 53

Figura 44 - Etapes del processat de la imatge 54
Figura 45 - Ús del Tesseract OCR 55
Figura 46 - Resultat del Tesseract OCR 55
Figura 47 - Icona de DroidText 55
Figura 48 - Diagrama de classes de l'aplicació 56
Figura 49 - Diagrama de seqüencia de l'aplicació 57
Figura 50 - Diagrama de seqüencia | Obtenir imatge 58
Figura 51 - Diagrama de seqüencia | Processat 59
Figura 52 - Format YUV 4:2:0 60
Figura 53 - El·lipse vs. “Bounding Box” 61
Figura 54 - Procés del OCR Tesseract en Android 62
Figura 55 - Obrir l'aplicació 63
Figura 56 - UI - Pantalla inicial 63
Figura 57 - UI - Utilització del zoom 64
Figura 58 - UI - Captura de la imatge 64
Figura 59 - UI - Processant la imatge 65
Figura 60 - UI - Resultats I 65
Figura 61 - UI - "Cropping" d'una imatge 66
Figura 62 - UI - Resultats II 66
Figura 63 - UI - Traducció 67
Figura 64 - UI - Enviar a 68
Figura 65 - UI - Enviar e-mail 68
Figura 66 - UI - Sel·lecció d'idioma 69
Figura 67 - Exemples d'imatges del set de proves 70
Figura 68 - Imatge amb reflex 71
Figura 69 - Imatge amb doble contrast 71
Figura 70 - Imatge texturada en excés 72
Figura 71 - Imatge desenfocada 72
Figura 72 - Caràcters continus 73
Figura 73 - Fals negatiu 73
Figura 74 - Caràcter erroni 74
Figura 75 - Diagrama de Gannt I | Desenvolupament 78
Figura 76 - Diagrama de Gannt II | Aplicació 78

Índex de taules

Taula 1 - "Confussion matrix" de dígits 15
Taula 2 - Evolució del mercat de "smartphones" segons SO 17
Taula 3 - "Confusion matrix" del classificador 50
Taula 4 - Característiques HTC Desire 56
Taula 5 - Resultats del test de l'algoritme 72
Taula 6 - Resultats del test del OCR 75
Taula 7 - Terminis del projecte 76
Taula 8 - Tasques fase de desenvolupament 76
Taula 9 - Tasques desenvolupament de l'aplicació 77
Taula 10 - Cost en recursos humans 79
Taula 11 - Cost del software 80
Taula 12 - Cost total del projecte 80

1. Introducció

En aquest document es presenta el desenvolupament d'una aplicació per a plataformes mòbils *Android* per a detectar zones amb text escrit i reconèixer aquest, en imatges obtingudes amb aquest mateix dispositiu mòbil.

1.1. Descripció del problema

El projecte tracta el reconeixement de text en imatges sobre qualsevol ambient. El principal objectiu és desenvolupar un algoritme que sigui capaç de detectar i aïllar les zones de text en una imatge extreta d'un dispositiu mòbil.

L'aplicació a desenvolupar serà capaç de millorar la imatge obtinguda per tal de extreure d'aquesta, totes les zones que no pertanyin al text, és a dir, d'una imatge com la de la *figura 1* n'obtindrem una altra de binaritzada que només contindrà text, com la *figura 2*.

Figura 1 - Imatge Original

Figura 2 - Imatge després del processat

Un cop obtinguda aquesta imatge, es podrà passar a un OCR que reconeixerà el text existent en la imatge, i així poder tractar-lo digitalment.

1.2. Restriccions al problema

A l'obtenir imatges directament des d'una càmera d'un dispositiu mòbil, no es pot assegurar que les condicions d'il·luminació o enfocament de la imatge seran les idònies per a detectar el text.

És per això, que el projecte es limitarà a la detecció en imatges en que el text sigui llegible i estigui íntegrament en la imatge, és a dir, no s'acceptarà aquelles imatges que continguin lletres escapçades, estiguin desenfocades o amb reflexos excessius, com es pot observar a la *figura 3*.

Figura 3 - Imatge amb reflexos

Altrament, l'ús d'un OCR imposa algunes restriccions extra. En primer lloc, la tipografia de la lletra a detectar no ha de ser lligada, ni basada en punts o traces no connexes. I en segon terme, l'OCR limita l'orientació del text, ja que només acceptarà text amb una petita rotació (menor de 25 graus).

Per últim el projecte es centrarà en zones on el text és de mida raonable, tipus senyals, o títols ja que en cas contrari, l'abast del projecte pot créixer en excés.

1.3. Aplicacions del projecte

Un cop obtingut el text en format digital, aquest pot ser usat en infinitat de situacions: editors de text, correus electrònics, traducció, etcètera. Per aquest projecte s'ha decidit, que el text podrà ser usat de tres maneres diferents.

Primer de tot, el text podrà ser enviat des de l'aplicació a d'altres programes o serveis, com xarxes socials , tipus Facebook o Twiter, possibilitant compartir amb altra gent qualsevol text que trobem en la nostra vida diària. També podrà ser enviat a traves de gestors d'arxius com Dropbox, i ser emmagatzemat en el núvol, per a futurs usos d'aquest. A més és podrà enviar el text reconegut per diferents protocols o aplicacions de comunicació com per exemple, missatges de text SMS, Bluetooth, Whatsapp o gestors de correu electrònic, permeten una integració total del text reconegut amb totes les aplicacions, que suporten text, del dispositiu mòbil.

En segon terme, l'aplicació pot ser de gran utilitat si s'està en un viatge fora de l'àrea de les llengües que l'usuari domina, o no té la fluïdesa suficient en aquestes. Amb aquesta aplicació, es reconeixerà el text, i un cop obtingut podrà ser traduït a gairebé totes les llengües del mon, fent senzilla la interpretació de senyals, o instruccions que es puguin necessitar en un d'aquests viatges.

Figura 4 - Icona Google Translate

Per últim, una de les grans utilitats de l'aplicació és el reconeixement de text amb accés universal. Persones amb deficiències visuals que tinguin dificultats en la lectura o inclús invidents podran utilitzar l'aplicació, el que podria ser una gran ajuda en la vida d'aquestes persones. Un cop reconegut el text, aquest podrà ser escoltat per l'usuari gracies a la utilització d'un sistema de text a veu, el qual narrarà el text que s'acaba de reconèixer. Per últim també pot ser d'utilitat en cas de persones que no saben llegir.

Figura 5 - Icona veu sobre text i accés universal

2. Conceptes previs

En aquest capítol es presenta un seguit de conceptes necessaris per a la realització del projecte.

En primer terme s'expliquen les bases necessàries sobre visió per computador que es necessitaran alhora de desenvolupar el projecte. En segon lloc es fa una breu introducció al sistema operatiu *Android*.

2.1. Fonaments de la visió per computador

En visió per computador les imatges s'acostumen a treballar en escala de grisos ja que, en la majoria dels casos, el color no és l'important de la informació que ens dona la imatge, sinó les formes i posicions dels objectes dins aquesta. Per això, el que s'acostuma a fer, és transformar les imatges amb 3 colors: vermell, verd i blau; a escala de grisos, és a dir, més o menys lluminositat.

Aquest procés es fa per a poder treballar de manera més ràpida i eficient sobre les imatges, ja que en comptes de tractar 3 valors per a cada píxel, un per cada color, només es treballarà amb un.

Figura 6 - Imatge en escala de grisos

Per les següents seccions, s'usarà la imatge de la *figura 6*, excepte que s'indiqui el contrari, com a entrada, per a mostrar les operacions i característiques bàsiques sobre les imatges, que s'usen en visió per computador.

2.1.1. Histogramació

L'histograma d'una imatge és una representació gràfica de la distribució dels valors de luminància de cada un dels píxels, d'una imatge en escala de grisos. Com es pot veure en la *figura 7*, en l'eix *'x'* s'expressen els valors de gris, i en l'eix *'y'* la freqüència d'aquests.

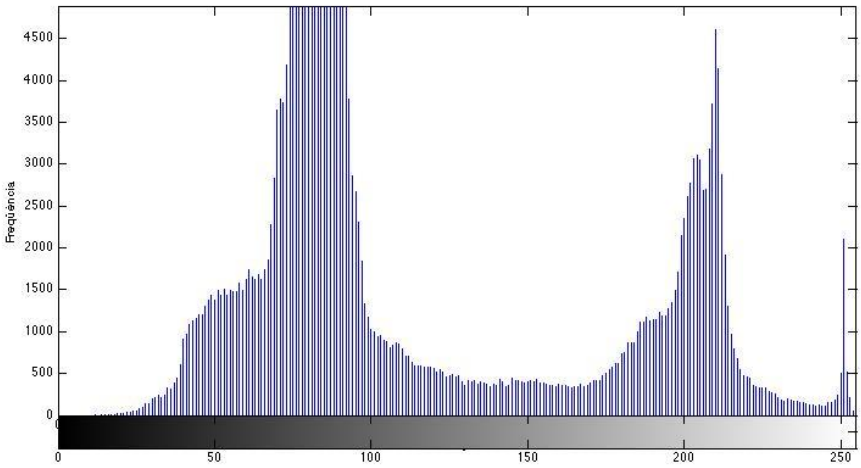


Figura 7 - Histograma de la imatge

Els histogrames s'utilitzen per a detectar característiques sobre les imatges com podrien ser, detecció d'imatges fosques, o clares (histogrames acumulats a la dreta/esquerra), diferents acumulacions de freqüència, etcètera.

2.1.2. Binarització

Binaritzar una imatge, és el procés que consisteix en dividir la imatge d'entrada en dos conjunts. S'assigna a cada píxel de la imatge, un nou valor, nivell alt o nivell baix (0 i 1). En el procés, per cada píxel de la imatge original, es compara el seu nivell de gris amb un llindar, i es decideix si es nivell alt o baix. Per posar un exemple, si s'agafa l'histograma de la *figura 7*, i s'intenta binaritzar, es podria adoptar com a llindar el valor 20 (per posar un cas) i per tant es tindrà una imatge amb gairebé tos els píxels a un nivell alt, és a dir tota blanca, ja que la freqüència de píxels per sota de 20, els quals identificaríem amb nivell baix, és a dir negre, es mínima.

Assignar un valor a aquest llindar és una feina complexa com es veurà a continuació amb la descripció d'alguns dels mètodes utilitzats per a aquest propòsit:

- ***Llindar global fix:***

El llindar (threshold en anglès), és prefixat a un nivell de gris. La imatge binaritzada *'F'*, doncs, és mol senzilla de calcular amb la següent formula:

$$F_{ij} = \begin{cases} 0 & si\ \ t < I_{ij} \\ 1 & altrament \end{cases}$$

On *'Fij'* i *'Iij'* corresponen al píxel *'i' 'j'* de la imatge binaritzada i de la imatge original, respectivament, i *'t'* és el llindar triat.

Aquest senzill mètode de binarització és clarament sensible a les variacions d'il·luminació de la mateixa imatge. En la *figura 8* es pot veure la imatge binaritzada amb diferents llindars. Es pot observar que la tria del llindar no es senzilla, ja que es pot generar soroll o fins i tot es pot donar el cas, de triar un llindar baix/alt i no veure correctament els objectes ($t = 64$). A més la tria no és automatitzable.

t = 64 t = 128 t = 192

Figura 8 - Binarització amb diferents llindars globals

- ***Llindar global flotant:***

Una millora a la solució anterior, és crear un llindar en funció de la imatge d'entrada. Això es pot aconseguir de diverses maneres fixant-se en diferents característiques de la imatge.

Una possibilitat és, calcular l'histograma i fixar un llindar tal que la meitat dels píxels estiguin per sobre el llindar i l'altra meitat per sota. Aquest mètode pot funcionar be en alguns casos com és el cas de la imatge que s'està tractant però en d'altres on la lluminositat varia en l'espai, pot comportar problemes amb el fons, com s'aprecia a la imatge central de la *figura 9*.

Les possibilitats per a calcular el llindar són extenses, alguns altres exemples podrien ser la mitja de determinats valors, o altres estadístiques similars.

- ***Llindar per regions:***

En comptes d'usar un llindar global per a tota la imatge, el que es pot fer és dividir-la en diferents regions, i binaritzar aquestes petites regions cada una per separat, això resta influencia als fons en que la il·luminació es va degradant, i dona una binarització més acurada.

La següent figura mostra la binarització amb llindar global o per regions:

Figura 9 - Binarització amb llindar global i per regions

2.1.3. Operacions morfològiques

Les operacions morfològiques són operacions, que es poden aplicar sobre les imatges, per tal de millorar o canviar alguna característica d'aquesta. En els següents punts es mostren alguns dels operadors basics usats en visió per computador, que seran necessaris per al projecte:

- ***Intersecció:***

La intersecció d'una imatge amb un altra imatge, també anomenada màscara, dona com a resultat una tercera imatge on els píxels actius d'aquesta segueixen la següent fórmula:

$$A \cap B \{p \quad si \quad p \in A \quad i \quad p \in B\}$$

És a dir tot aquell píxel *'p'* que pertanyi alhora a la imatge *'A'* i a la *'B'* pertany a la imatge intersecció. En la següent figura es mostra la intersecció de la imatge original binaritzada amb una màscara formada per una sola franja de píxels.

Màscara Imatge resultat

Figura 10 - Intersecció d'imatges

- ***Unió:***

La unió de dos imatges, és una operació similar a la intersecció però es regeix per una fórmula diferent:

$$A \cup B \{p \quad si \quad p \in A \quad o \quad p \in B\}$$

Per tant la imatge unió és aquella tal que, tot píxel *'p'* de cada imatge és en la imatge final. La següent figura en mostra un exemple amb la mateixa imatge i màscara:

Màscara Imatge resultat

Figura 11 - Unió d'imatges

- ***Complement:***

El complement, o també anomenat negatiu, d'una imatge és una operació que segueix la següent fórmula:

$$\bar{A} \{p \quad si \quad p \notin A\}$$

És a dir, que la imatge resultant és aquella que conte actius tots els píxels *'p'* que la imatge original no tenia i inactius els altres. En la següent figura es mostra un exemple de la imatge que estem tractant:

Imatge original Imatge complement

Figura 12 - Complement d'una imatge

- ***Erosió:***

L'erosió d'una imatge, és una operació morfològica en la qual, cada píxel inactiu de la imatge original es estès per una petita imatge binaria, anomenada element estructurant. És a dir, si es té com a element estructurant, un quadrat amb 3 píxels de costat, centrat en l'origen, i s'aplica una erosió sobre una imatge, tots els píxels inactius d'aquesta hauran absorbit els veïns. Absorbir els veïns vol dir que, si els veïns ja eren inactius no tenim cap alteració, en canvi, si el veí era actiu aquest passarà a ser inactiu. Per tant, en aquest exemple, tots els píxels "frontera" entre actiu i inactiu, han passat a esser inactius.

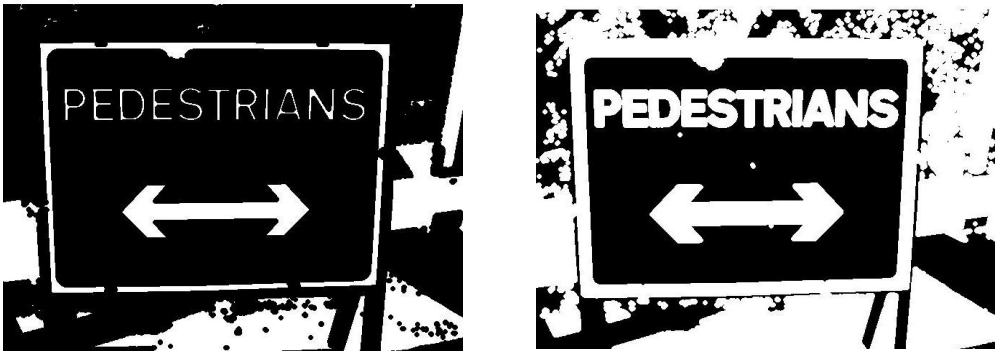
La següent imatge mostra l'erosió de la imatge original binaritzada amb un element estructurant en forma de disc:

Imatge original Imatge erosionada

Figura 13 - Erosió d'una imatge

- ***Dilatació:***

La dilatació és l'operació oposada a la erosió, és a dir, que cada píxel actiu de la imatge original, es estès segons la forma de l'element estructurant. La següent figura mostra la erosió i dilatació de la imatge original binaritzada, amb el mateix element estructurant que s'ha usat en la *figura 13*.



Erosió Dilatació

Figura 14 - Dilatació d'una imatge

- ***Opening:***

El "opening" d'una imatge és una operació composta, que segueix la següent fórmula:

$$F = (A \ominus B) \oplus B$$

essent *'F'* la imatge resultant, *'A'* la imatge original, *'B'* l'element estructurant, i $\ominus$, $\oplus$ l'operador erosió i dilatació respectivament. Per tan el "opening" és l'aplicació d'una erosió, seguida per una dilatació, ambdues amb el mateix element estructurant.

Aquest operador és usat majoritàriament, per a eliminar soroll de la imatge, com podrien ser petits objectes en el fons. La següent figura mostra un detall de la imatge que s'està tractant abans i després del operador:

Figura 15 - Opening d'una imatge

- ***Closing:***

El "closing" d'una imatge també és una operació composta, pràcticament oposada al "opening", que segueix la següent formula:

$$F = (A \oplus B) \ominus B$$

essent *'F'* la imatge resultant, *'A'* la imatge original, *'B'* l'element estructurant, i $\ominus$, $\oplus$ l'operador erosió i dilatació respectivament. És a dir, aplicar una dilatació per tot seguit aplicar una erosió amb el mateix element estructurant.

L'operador "closing", es sol utilitzar per a omplir petits forats que hagin pogut quedar després de la binarització, en els objectes de l'escena. La següent figura mostra un detall de la imatge que s'està tractant abans i després del operador:

Figura 16 - Closing d'una imatge

2.1.4. Etiquetatge

Un cop obtinguda una imatge binaritzada, es vol extreure característiques dels objectes que l'escena de la imatge, conté. Per a poder esbrinar les característiques dels objectes, primer s'haurà de tenir una llista d'aquest objectes. Aquesta llista s'obtindrà amb un procés anomenat etiquetatge de components connexos, que s'explica tot seguit.

L'etiquetatge dels components connexos d'una imatge és el procés consstent en dividir una imatge en diversos grups de píxels. Tots els píxels que formen un grup són adjacents entre ells, és a dir, són un objecte per ells mateixos. Per a decidir si dos píxels són adjacents, hi ha bàsicament dos opcions a considerar, les quals seran més o menys adients, segons l'aplicació que s'estigui dissenyant.

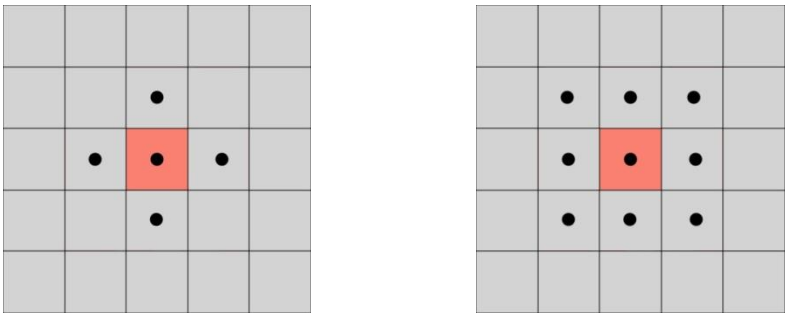


Figura 17 - Connectivitat 4 i 8 d'un píxel

- *Connectivitat 4*: Un píxel és adjacent a un segon píxel, si aquest es troba a distancia 1 en qualsevol direcció horitzontal o vertical. La imatge esquerra de la *figura 17* mostra els píxels adjacents (punts) al píxel original (vermell) segons la connectivitat 4.

- *Connectivitat 8*: Un píxels és adjacent a un altre si és adjacent en connectivitat 4 o si és troba a distancia 1 en direcció diagonal. La imatge dreta de la *figura 17* mostra els píxels adjacents (punts) al píxel original (vermell) segons la connectivitat 8.

En la següent figura es mostra l'etiquetatge de la imatge que s'està tractant, cada component connex (CC) etiquetat, ha estat pintat amb un color diferent.

Figura 18 - Imatge etiquetada

2.1.5. Extracció de característiques

Un cop obtinguda una llista amb tots els CC de la imatge, es poden obtenir característiques sobre ells, tals com, l'àrea que ocupen, la distribució, l'orientació, entre d'altres.

Una característica molt usada és la *'Bounding Box'* d'un objecte. La *'Bounding Box'*, és el rectangle més petit que conté tot el component connex. Aquest rectangle té diverses utilitats com es veurà en el desenvolupament del projecte. La següent figura mostra la *'Bounding Box'* d'alguns dels CCs de la imatge original.

Figura 19 - Bounding Box d'un cc

2.1.6. Classificació de regions

Un cop s'és capaç d'obtenir diferents característiques, es voldrà classificar cada un dels components connexos, per a poder ser capaços de reconèixer quin tipus de regió s'està tractant. En els següents punts es descriu alguns dels mètodes que es poden usar per classificar regions.

- ***Classe més pròxima:***

Aquest mètode es basa en comparar els diversos CCs o regions, amb diferents classes prefixades, i etiquetar cada una de les regions o CCs amb el nom de la classe a menys distancia.

Per a això abans de poder classificar, s'haurà de crear classes model, és a dir, classes de CCs que es diferenciïn clarament entre elles, i que dins elles comparteixen característiques en comú. Es tindrà una classe per a cada tipus de CC que pugui aparèixer, la qual serà representada per un o més vectors de característiques.

Aquests vectors contindran a cada posició, el valor d'una característica de la regió que el diferencii d'altres tipus de CC. Cada vector diferent dins la classe, representarà una regió del mateix tipus.

Per a cada regió que es vulgui classificar s'haurà de crear un vector de característiques, amb les mateixes característiques ja triades per a definir les classes. Així en aquest moment es podrà comparar el vector ara generat amb cada una de les classes model.

En primer terme, es necessita un mètode per a comprovar com de semblants són dos vectors de característiques. El mètode més senzill és la distància normal entre dos vectors, que es representa per la següent formula:

$$Distancia = \sum_{i=1..N} |a[i] - b[i]|$$

essent *'a'* i *'b'* els dos vectors de característiques a comparar. Altres mètodes més complexes, poden ser la distancia Euclidia, la distancia Mahalanobis, o altres de mes senzilles com la distancia discreta, que només té en compte el nombre de igualtats entre vectors.

Ara que es tenen les classes model i les regions a tractar, ja caracteritzades, i es té un mètode per a comparar-les, existeixen diversos mètodes per tal d'assignar el CC a una classe o un altra.

El primer i més senzill, es comparar el vector de característiques de cada regió amb cadascun dels vectors de les classes model, i assignar a la regió la etiqueta del més pròxim. Aquest mètode es potencialment costós en temps, si es té molts models per a cada tipus. Una solució al problema del temps, podria ser, fer la mitja dels vectors de característiques dins d'una mateixa classe, per a reduir el nombre de vectors a comparar amb cada component.

Un altra millora al mètode podria ser agafar, no només la classe més pròxima, sinó les *'n'* més pròximes i buscar la moda entre elles. Per acabar, una millora en l'eficiència és, crear un classificador heterogeni, és a dir, que cada característica no és igual, n'hi ha que són més valuoses que d'altres i per tant s'han de ponderar per a obtenir bons resultats.

- ***Arbres de decisió:***

Una altra solució alhora de classificar regions o components connexos, són els arbres de decisió. Un arbre realitza decisions sobre cada característica, amb un ordre específic, és a dir, en cada node del arbre es fa una decisió basant-se en una característica diferent. Les fulles del arbre són les classes, que poden ser repetides en varies fulles.

Començant des de l'arrel, es van prenent decisions segons les característiques de la regió, fins a arribar a una fulla en la qual la regió serà etiquetada amb el nom d'aquella classe.

Els arbres de decisió poden ser apresos o entrenats amb dades del problema, i així ser creats de forma automàtica.

Altres mètodes per a classificar podrien ser xarxes neuronals, regles, etcètera. Per a avaluar si el mètode escollit per al classificador funciona amb eficàcia, podem comprovar la "*confussion matrix*".

La "*confussion matrix*" és una matriu quadrada, que tant en files com en columnes, s'hi llista cada una de les classes del problema. Les files representen l'etiqueta real de la regió i les columnes en representen el valor adjudicat pel classificador.

En un classificador perfecte, aquesta matriu, només a de tenir valors en la diagonal. Si conté valors per sobre o per sota d'aquesta diagonal, ens mostra, que el classificador etiqueta malament certes regions, quant més nombres són fora de la diagonal, menys eficient és el classificador. Aquestes matrius també es poden representar en forma de taules com es mostra tot seguit, en un exemple d'un classificador de dígits. Com es pot veure el classificador confon el dígit 6 amb el dígit 9 i viceversa, una de les causes podria ser que aquests dígits són simètrics l'un respecte l'altre.

	Zero	Un	Dos	Tres	Quatre	Cinc	Sis	Set	Buit	Nou
Zero	12	0	0	0	0	0	0	0	0	0
Un	0	15	0	0	0	0	0	0	0	0
Dos	0	0	21	0	0	0	0	0	0	0
Tres	0	0	0	13	0	0	0	0	0	0
Quatre	0	0	0	0	17	0	0	0	0	0
Cinc	0	0	0	0	0	32	0	0	0	0
Sis	0	0	0	0	0	0	10	0	0	13
Set	0	0	0	0	0	0	0	19	0	0
Buit	0	0	0	0	0	0	0	0	11	0
Nou	0	0	0	0	0	0	7	0	0	15

Taula 1 - "Confussion matrix" de dígits

2.2. Sistema Android

Android és un sistema operatiu per a dispositius mòbils, tals com "smartphones" o "tablets", basat en Linux i desenvolupat per la *Open Handset Alliance*, alliberat sota la llicencia *Apache v2.0*. El consorci liderat per *Google*, està format per varies empreses del entorn dels dispositius mòbils, com ara *HTC, Sony, Dell, Intel, Motorola, Texas Instruments, T-Mobile,* entre d'altres, i és dedica al desenvolupament d'estàndards oberts relacionats amb els dispositius mòbils.

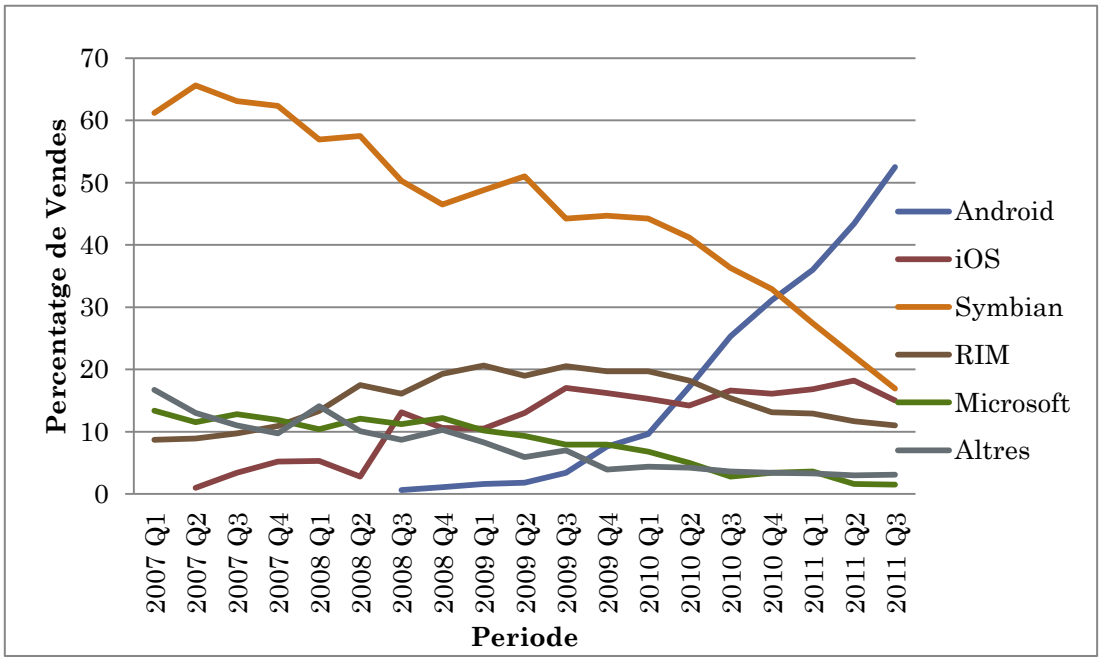
2.2.1. Historia del sistema

Android, Inc. va ser fundada el 2003 a California, Estats Units, per Andy Rubin, Rich Miner, Nick Sears i Chris White, tots en el sector de la telefonia i els dispositius mòbils, per a desenvolupar telèfons intel·ligents. Al 2005 *Google* va voler entrar al sector dels dispositius mòbils, comprant *Android, Inc.*, a més els seus desenvolupadors més importants, van passar a formar part de les files del gegant de la informàtica.

A finals del 2007 el consorci *Open Handset Alliance* presentà, el mateix dia de la seva formació, el seu primer producte, el projecte Android, un sistema operatiu basat en el kernel de Linux 2.6.

El projecte de codi Obert Android(AOSP), liderat per *Google*, és l'encarregat de mantenir i desenvolupar el sistema *Android*, així com, assegurar la compatibilitat dels dispositius, el sistema, i les aplicacions que puguin ser dissenyades per tercers.

El primer dispositiu amb *Android* que va sortir al mercat, als voltants de 2008, és el *HTC Dream*. Des de llavors fins ara, el projecte no ha parat de créixer, tant en funcionalitats, dispositius, i usuaris, com podem veure en la *taula 2*, on actualment *Android* és el sistema operatiu per a dispositius mòbils més venut, per davant de *Symbian*, usat majoritàriament en dispositius *Nokia*, *iOS* d'*Apple* i *RIM* fabricant dels dispositius mòbils coneguts com *BlackBerry*.[1]



Taula 2 - Evolució del mercat de "smartphones" segons SO

2.2.2. Evolució de les versions

Des de finals de 2008 fins a l'actualitat *Android* no a parat d'afegir funcionalitats al SO en les seves diferents versions, fins l'actual *'Ice cream sandwich'*, les diferents millores són mostrades a continuació.

- ***Android 1.0:***

Al setembre de 2008 la primera versió d'*Android* va sortir al mercat amb el telèfon intel·ligent *HTC Dream*.

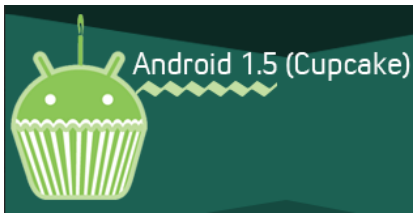
Les característiques principals del sistema eren:

- Integració amb els serveis de *Google*
- Navegador web capaç de mostrar múltiples pagines en HTML i XHTML
- Creació d'un mercat d'aplicacions i actualitzacions
- Possibilitat d'executar varies aplicacions simultànies amb 'Multitasking'
- Missatgeria instantània
- Integració de la càmera
- Wi-Fi i Bluetooth

A principis de 2009 va sortir al mercat una actualització del sistema, la 1.1 amb algunes millores menors.

- ***Android 1.5 'Cupcake':***

Pels vols d'Abril del 2009 *Android* ja tenia un 3% del mercat de "smartphones" mundial, i va llançar la nova versió del sistema basat en el kernel de Linux 2.6.27.

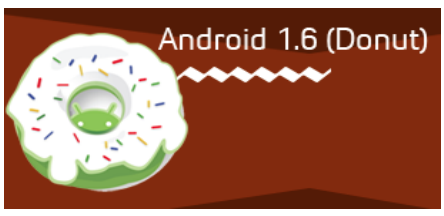


Les principals millores del sistema *'Cupcake'* foren:

- Millora de l'adquisició i rapidesa de la càmera
- Millora en l'adquisició del GPS i inclusió del AGPS
- Introducció del teclat virtual en pantalla
- Integració amb *YouTube* i *Picassa*

- ***Android 1.6 'Donut':***

Quatre mesos més tard que la versió 1.5, la nova versió *'Donut'* va ser alliberada basada en el kernel de Linux 2.6.29.



Algunes de les millores que incorporava eren:

- Integració de la cerca, i cerca per veu
- Creació de la galeria d'imatges
- Integració de captura de vídeo
- Indicador de l'ús de la bateria
- Suport per a xarxes CDMA
- Funció de text a veu multi-llenguatge

- ***Android 2.0 Eclair':***

Amb el mateix kernel de Linux i amb només un més de diferencia respecte la versió 1.6, la versió *'Eclair'* va sortir a finals d'Octubre del 2009.

Les principals característiques que diferencien la versió són:

- Possibilitat de múltiples comptes de e-mail
- Sincronització de contactes
- Suport per a Bluetooth 2.1
- Suport de HTML5 per al navegador
- Noves característiques als calendaris

En els següents dos mesos dos noves actualitzacions van sortir a la llum 2.0.1 al desembre i 2.1 a principis del següent any. Aquestes incorporaven petites millores i actualitzacions a un sistema que ja tenia molta acollida al mercat mundial.

- ***Android 2.2 'Froyo':***

A finals de maig de 2010 amb gairebé un 20% del mercat, superant a *Microsoft* i igualant a *Apple*, va ser alliberat *'Froyo'*, basat en el kernel de Linux 2.6.32.

Les principals innovacions de la versió foren:

- Inclusió d'*Adobe Flash* 10.1
- Millora del suport de *Exchange*
- Suport per a Wi-Fi Hotspot
- Suport per a múltiples teclats i llenguatges
- Suport per a dispositius amb Bluetooth (com ara automòbils)

Les versions 2.2.1 i 2.2.2 van ser alliberades per a solucionar diversos bugs a principis del 2011. Finalment la versió 2.2.3 va ser llançada al novembre del mateix any amb dos nous paquets de seguretat.

- ***Android 2.3 'Gingerbread':***

'Gingerbread' va ser alliberat als inicis de desembre de 2010 quan ja més del 30% de mercat de "smartphones" era dominat per *Android*, només *Symbian* plantava cara al nou sistema tot i que amb una gran davallada.

Les millores incorporades a la nova versió foren:

- Millores de la interfície en quant a rapidesa i senzillesa
- Nova distribució del teclat virtual
- Selecció, copiat i enganxat de text
- Suport per a *'Near Field Comunication'*
- Suport per a veu sobre IP
- Suport per a múltiples càmeres i nous sensors

Les versions 3,4,5,6,7 de la versió 2.3 van ser alliberades entre principis i d'any i setembre del mateix afegint noves funcionalitats, com suport de veu i vídeo per a *GoogleTalk*, així com solució a diversos errors.

- ***Android 3.0 'Honeycomb':***

Basat en el kernel de Linux 2.6.36, *Android* feia un pas més enllà dels "smartphones", per atacar el mercat de les "tablets", amb la versió *'Honeycomb*' del sistema, durant febrer de 2011.

Entre d'altres millores, les característiques de la versió eren:

- Optimització especial per a tablets i dispositius amb pantalles grans
- Tethering mitjançant Bluetooth
- Suport per protocols de transferència de Imatges/vídeos
- Millora al multitasking, notificacions, widgets i home personalitzable
- Inclusió de l'acceleració hardware
- Suport per processadors de múltiples nuclis

Durant els següents mesos diferents versions de *'Honeycomb'* sortiren a la llum, aportant petites funcionalitats com ara suport per teclats externs o joysticks.

- ***Android 4.0 'Ice cream Sandwich':***

L'ultima versió fins el moment d'*Android*, *'Ice cream Sandwich'* va ser alliberada l'octubre del 2011 basada en el kernel de Linux 3.0.1.

Apart de les millores en rendiment i funcionalitats existents, algunes de les característiques pròpies de la versió són:

- Reemplaçament dels botons físics per virtuals
- Sistema de reconeixement facial
- Ampliació de les funcionalitats de la càmera
- Gravació de vídeo 1080p

Actualment l'ultima versió del sistema és la 4.0.4 alliberada el Febrer de 2012.

2.2.3. Estructura del sistema operatiu

Android està estructurat en diferents capes. Com a nivell més basic hi ha un nucli basat en el kernel de Linux, més amunt hi ha un middleware, algunes llibreries i APIs escrites en C, a més conté un framework d'aplicacio i algunes llibreries escrites o compatibles amb *Java*. *Android* usa la maquina virtual *Dalvik* per a compilar en temps real les aplicacions un cop traduïdes de Java a Dex-code (executables de *Dalvik*).

Respecte al hardware el sistema operatiu esta dissenyat bàsicament per a arquitectures ARM, tot i que té suport per a arquitectures x86 mitjançant el projecte Android x86.

En la següent figura podem veure un esquema de les principals característiques d'*Android*:

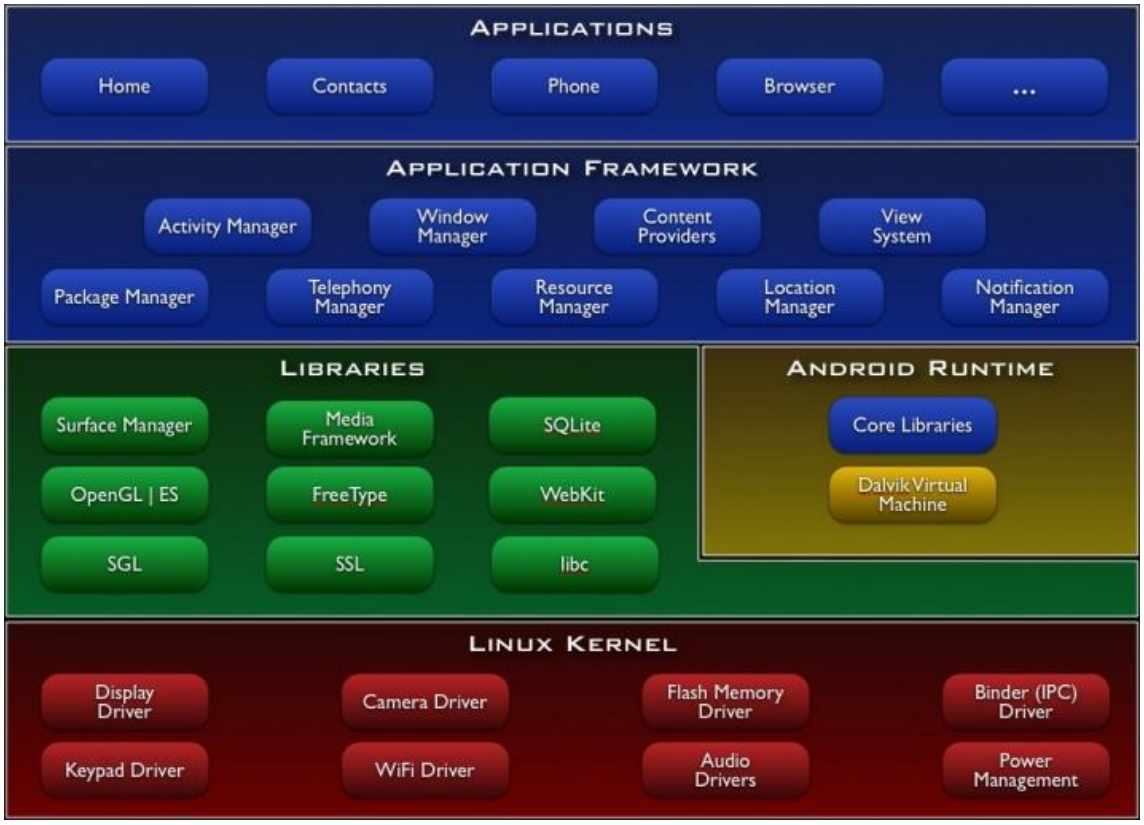


Figura 20 - Arquitectura del sistema Android

Com podem veure el kernel s'encarrega de tenir una capa d'abstracció entre el hardware, que és operat per mitjà dels diferents drivers, i la resta del software.

La *Android Runtime* és l'encarregada de proveir les llibreries amb la majoria de funcionalitats bàsiques del nucli de Java. Cada aplicació que s'executa en el sistema té el seu propi procés, amb una instancia de la maquina virtual *Dalvik* pròpia per a cada un d'ells.

Les llibreries incloses, estan escrites en C/C++ i són proposades als desenvolupadors a través del framework d'aplicació. Entre d'altres, les llibreries ofereixen accés a la interfície d'usuari, codificació i reproducció d'àudio i vídeo, el motor del navegador web, llibreries 3D o accés a bases de dades.

Com ja s'ha dit el framework d'aplicació proporciona al desenvolupador l'habilitat d'utilitzar el hardware del dispositiu per a qualsevol acció, sempre que estigui disponible, com pot ser, obtenir la informació de localització, engegar processos en segon pla, afegir alarmes, entre d'altres moltes funcionalitats.

Això permet al desenvolupador usar la seva imaginació per a construir qualsevol aplicació, creant components, o aprofitant components o aplicacions ja creades per altres usuaris i afegir-les a la seva aplicació, tot de forma senzilla, i amb un llenguatge standard i orientat a objectes, com és Java.

Una aplicació *Android* esta dividida en varies parts que s'expliquen a continuació:

- ***Codi Java:***
  Classes i funcions escrites en Java que donen la funcionalitat a l'aplicació. Més internament podem trobar Activitats i Serveis com a components més importants. Les activitats representen una pantalla amb interfície d'usuari, i els serveis, són processos que poden córrer en segon terme. Els components poden ser mostrats des d'altres components, per així crear la línia de treball de l'aplicació.

- ***Recursos:***
  Les imatges i vídeos que necessiti l'aplicació formaran part d'aquesta secció, però a més també i tindrem qualsevol objecte relacionat amb la representació visual de la nostra aplicació, ja siguin els fitxers XML amb la representació de la interfície o fitxers que continguin el text d'aquesta.

- ***Arxiu de manifest:***
  Es tracta d'un arxiu XML on es declara els components que té l'aplicació, els permisos que necessita l'aplicació, és a dir, quines funcionalitats farà servir (internet, GPS, llegir/escriure de memòria externa, etc.), també s'hi especifica els requisits de l'aplicació, és a dir, el hardware que fa servir (càmera, etc.) o el nivell mínim de SDK requerit per a executar totes les funcions de l'aplicació.

3. Estat de l'art

En els últims anys el reconeixement de text ha tingut una gran repercussió degut a la intenció de digitalitzar documents automàticament, des de vells manuscrits a conservar, fins a petits documents impresos dels quals en volem copia digital, o qualsevol tipus de documents que es vulguin arxivar per a consultar més tard.

Les aplicacions que tracten de reconèixer text s'anomenen Reconeixement Òptic de Caràcters (OCR) del anglès *Optical Character Recognition*. Un OCR extreu d'una imatge, els caràcters que composen un text, per a emmagatzemar-los en un format el qual pugui interactuar amb programes d'edició de text.

Mentre en una imatge els caràcters són descrits per cada un dels píxels que forma la imatge, al convertir-los en format de text (per exemple ASCII o Unicode), passen a ocupar només un nombre, el qual els identifica, amb una reducció significativa d'espai en memòria. Un cop convertit es pot utilitzar aquest text per a una infinitat d'aplicacions, tals com traducció o aplicacions de text a veu.

En el següent apartat es mostren diverses solucions comercials a aquest problema.

3.1. Aplicacions actuals

La majoria d'aplicacions actuals que hi ha al mercat, estan enfocades a la detecció de text en documents escanejats, és a dir, imatges mínimament controlades. Es poden dividir els sistemes actuals de reconeixement en dos grups:

- ***Reconeixement de text en PC:***

  Els sistemes de reconeixement per a ordinadors es basen únicament en detecció de text en documents escanejats, clarament contrastats i amb entorns poc o molt controlats. Els sistemes de reconeixement més usats són els següents:

  - *OmniPage* [2] : És una aplicació comercial dedicada al reconeixement de text a gran escala, és a dir, grans volums de dades. Té un cost aproximat d'uns 400€.

  - *ABBYY FineReader* [3] : És tracta d'un sistema de reconeixement de text en documents escanejats o d'imatges de documents. Té un cost aproximat de 180€.

- *Readiris* [4] : És un altre sistema de reconeixement de text en documents exclusivament. Té un cost aproximat de 130€ la versió bàsica o 500€ una versió de més qualitat.

- ***Reconeixement de text en Mòbil:***

Aquest tipus d'aplicacions han sorgit en els últims anys gracies als anomenats "smartphones", ja que tenen una capacitat de càlcul comparable a un ordinador. És una part del mercat poc explotada i per tant i trobem molt poques aplicacions les quals es poden llistar a continuació:

- *Prizmo* [5] : És una aplicació per a *iPhone* que reconeix text escrit i el pot traduir. Està orientat a documents, targetes de contacte o tiquets de compra.

Figura 21 - Prizmo iPhone App

- *Word Lens* [6]: És una altra aplicació per *iPhone* que tradueix, entre espanyol i angles, i mostra per pantalla el text reconegut.

Figura 22 - Word Lens iPhone App

- *Google Goggles* [7]: És una aplicació per a *Android* o *iPhone* capaç de reconèixer objectes i text en imatges. El processat de les imatges no es fa en el dispositiu mòbil sinó que es realitza en els servidors de *Google*, a diferencia de les altres aplicacions.

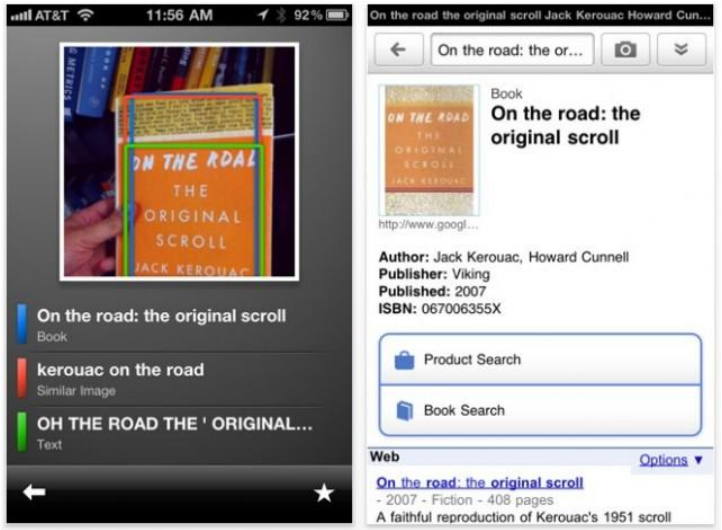


Figura 23 - Goggles iPhone App

Com es pot apreciar en les figures del reconeixement mòbil, com per exemple la *figura 22*, on el text detectat no és ni la meitat del text que podem apreciar a la imatge, el ratio d'encert ha disminuït notablement ja que tant la qualitat de la imatge com l'entorn deixen de estar controlats.

3.2. Algoritmes de segmentació de la imatge

Com ja s'ha vist anteriorment, és possible que en les imatges obtingudes no s'hi trobi un fons clarament diferenciat dels caràcters, per a detectar on és el text, com podria ser la vegetació. Per aquesta raó es necessita un preprocés de la imatge abans de carregar-la en un OCR. Aquest processat ha de permetre extreure de la imatge totes aquelles parts que no pertanyen a regions amb caràcters.

En els següents apartats es pot veure diferents treballs que descriuen mètodes de segmentació.

3.2.1. Segmentació usant 'Toggle-mapping'

En el treball [8], presenten un mètode basat en l'ús d'un operador morfològic anomenat *Toggle-mapping*. Aquest operador, donada una funció f i un conjunt de funcions h_1 .. h_n , una nova funció k es definida de la següent forma:

$$\forall x \in D_f \; k(x) = h_i(x); \;\; \forall j \in \{1 \,..\, n\} \;\; |f(x) - h_i(x)| \leq \left|f(x) - h_j(x)\right|$$

Aquest algoritme fa servir aquest operador morfològic per a segmentar la imatge (en escala de grisos). En concret fa servir com a conjunt de funcions h_1 .. h_n dues funcions. Aquestes són els operadors morfològics erosió i dilatació aplicats sobre la imatge f, les quals són calculades de la següent forma:

$$\forall x \in D_f \; h_1(x) = \min f(y); y \in v(x)$$
$$\forall x \in D_f \; h_2(x) = \max f(y); y \in v(x)$$

essent $v(x)$ el veïns del píxel x segons l'estructura de l'operador.

En comptes d'agafar directament la funció k del *toggle-mapping*, l'algoritme en fa una adaptació per a reduir el soroll "salt" i "peeper" que aquest genera:

$$\forall x \in D_f \; s(x) = \begin{cases} 0 & Si \; |h_1(x) - h_2(x)| < C_{min} \\ 1 & Si \; |h_1(x) - h_2(x)| \geq C_{min} \\ & \& \; |h_1(x) - f(x)| < p * |h_2(x) - f(x)| \\ 2 & altrament \end{cases}$$

essent $s(x)$ la nova imatge generada, C_{min} el mínim contrast a detectar i p una variable per controlar el gruix de les estructures detectades. Un cop aplicat aquest operador la imatge no és binaria sinó que consta de 3 valors, un d'alt, un de baix i un que representa les zones homogènies. Per a binaritzar la imatge, és a dir, tenir 2 valors en comptes de 3, aquest algoritme proposa estudiar els límits de les regions petites per tal d'omplir un possible buit dins els caràcters seguit d'un petita dilatació.

3.2.2. Stroke width transform

Aquest mètode, descrit en el treball [9], tracta la reconstrucció de les traces de les lletres per després reagrupar els píxels, primer en lletres, i més tard en paraules. Per aconseguir-ho fa servir el que han anomenat *'Stroke Width Transmorm'*, un operador local que calcula per cada píxel, l'amplada de la traça, a la que més provablement pertany. Així, retorna una matriu, amb mida igual a la de la imatge, on cada píxel o posició conté, l'amplada estimada de la traça a la que pertany aquest píxel.

Per a poder reconstruir les traces, primer es calculen els contorns aplicant l'operador *'Canny'* [10]. Un cop fet això, es calcula el gradient de cada un dels píxels *p*. Si *p* està en el contorn d'una traça, el gradient ha de ser, aproximadament, perpendicular a la direcció d'aquesta traça. Llavors, es pot generar un raig en la direcció del gradient i seguir-lo fins a trobar un nou píxel *q*. Si el gradient d'aquest, es oposat al gradient del píxel *p,* cada un dels píxels entre *p* i *q* passarà a tenir un nou valor. Aquest valor serà, la distancia entre *p* i *q* a no ser que ja tingui un valor menor, cas en el qual es deixarà el valor anterior.

Un cop es té tots els píxels anotats, i per tant la matriu ja formada, s'agruparan aquells veïns que tinguin una amplada de traça similar. Amb això s'aconseguirà tenir una imatge dividida en diverses regions, és a dir, segmentada.

3.2.3. Ultimate Opening

En el treball [11], s'investiga sobre l'operador *'Ultimate Opening'*. Aquest és un operador residual que es basa en observar els residus en successives transformacions de la imatge. Cada píxel és observat després d'aplicar diversos 'open' a la imatge, amb diferents mides d'element estructurant. La informació a tenir en compte és la diferencia entre dos 'open', que anomenarem residu. Per a cada píxel es guarda el màxim valor dels diferents residus que s'estan obtenint.

Per a extreure els components connexos d'aquesta "imatge residual" s'apliquen les següents tècniques:

- "Low threshold" per a poder eliminar components amb molt poc contrast.
- "Local adaptative threshold" a cada component per a separar possibles components que hagin pogut ser mesclats. Aquest llindar és calculat amb la moda de la imatge residual.
- S'ha d'aplicar dues vegades l'operador, sobre la imatge i sobre la seva inversa, per a poder detectar, fosc sobre clar i viceversa.

Així s'haurà extret tots els components connexos de la imatge.

3.2.4. Localised Measures

Un altre mètode per a segmentar la imatge és l'explicat al treball [12]. Aquest mètode es basa en les diferencies estadístiques de les regions amb text. Per a separar aquestes àrees l'algoritme fa servir 5 mesures estadístiques:

- M_1 La variància de l'histograma d'escala de grisos dins el cercle de veïns.
- M_2 La densitat de contorns (calculats amb l'operador *'Sobel'*) en un cercle de veïns
- M_3 La diferencia entre el histograma del píxel i els seus veïns (8 píxels)
- M_4 La asimetria dels contorns en un cercle de veïns
- M_5 La dispersió dels contorns sobre totes les direccions

Per a combinar les mesures i obtenir la imatge binaria, l'algoritme fa us d'una xarxa neuronal de 3 capes on les entrades són les 5 mesures i la sortida és una imatge separada en zones de text i soroll.

3.3. Detecció de text en imatges segmentades

Un cop s'ha obtingut la imatge segmentada, s'ha de decidir quines regions són text i quines no. Per això, en els següents apartats es mostren diversos mètodes de filtrat i classificació de regions.

3.3.1. Filtrat de regions

Les zones amb textura, acostumen a generar molts components connexos. La missió de les següents regles és, filtrar moltes d'elles, d'una manera rapida, per a que ens pugui estalviar temps a l'hora de classificar els caràcters:

- Els caràcters estan formats per traces de gruix constant. S'estima el gruix de cada component connex i s'eliminen aquells que la seva alçada, és dos cops més petita que el seu gruix. El gruix pot ser estimat amb les funcions de distancia en les direccions horitzontal i vertical.

- Els caràcters han de ser visibles. És descarten totes aquelles regions que la seva àrea és massa petita per a ser un caràcter.

- Els caràcters no ocupen tota la imatge. És descarten aquelles regions que ocupin molt espai de la imatge.

3.3.2. Detecció de caràcters

Un cop s'ha reduït el nombre de regions a tractar s'ha de centrar-se en classificar els components connexos restants, en text o soroll basant-se en les següents característiques:

- ***Característiques geomètriques:***
  Com poden ser l'alçada i l'amplada de la *'Bounding Box'*, l'àrea de la regió o l'àrea de la *'BoundingBox'*. Amb les inverses d'aquestes també es pot calcular diferents *'aspect ratio*'. Aquestes característiques i les seves combinacions són eficaces per a extreure regions que no són text però la seva eficàcia disminueix en entorns no controlats.

- ***Estimació del gruix de la traça:***
  Com ja s'ha vist abans el gruix d'una traça pot ajudar a decidir si s'està tractant un caràcter o no. Aquí es proposa estimar el gruix de maneres més costoses en temps d'execució com podrien ser; el màxim de la funció distancia dins del component connex, o la distribució dels píxels dins el component connex, entre d'altres.

- ***Regularitat de la forma:***
  Els caràcters tendeixen a tenir una forma més regular que no pas regions arbitraries, un nombre de forats limitat, contorns regulars i una important compacitat. És podria usar, el nombre d'*Euler*, el perímetre, la compacitat o la complexitat entre d'altres per a classificar correctament les regions.

- ***Contrast:***
  Les regions amb text haurien d'esser contrastades per a poder llegir-les amb claredat, això dona un altra característica a tenir en compte. Per a estimar el contrast d'un component connex es pot calcular la seva màxima variància intraclase en la seva *'BoundingBox'*.

Amb les regions ja caracteritzades, es necessita un algoritme que classifiqui els diferents components connexos en text o soroll. Per aquesta tasca es pot usar qualsevol dels mètodes de classificació explicats en la *secció 2.1.5*, com poden ser, arbres de decisió, xarxes neuronals, distancies, etcètera. Un cop classificades totes les regions s'hauran de conservar només aquelles que han estat etiquetades com a text.

3.4. Reconeixement de text(OCR)

Un cop la imatge està correctament tractada, és a dir; segmentada , filtrada i etiquetada, es pot donar-la com a entrada a un OCR. Qualsevol de les aplicacions mostrades en el *capítol 3.1* podrien servir per al reconeixement dels caràcters, però, al contrari que els següents sistemes, són aplicacions finals destinades als usuaris i que per tant no es podrien fer servir a l'hora de desenvolupar o traslladar a un altre sistema o llenguatge.

En els següents punts es poden veure 3 dels sistemes de lliure distribució més usats en l'actualitat:

- *GOCR* [14]: És una aplicació desenvolupada sota *GNU Public License*, converteix imatges de text escanejades en arxius de text. Té aproximadament un 88% de precisió en el reconeixement.

- *Ocrad* [15]: També és una aplicació sota *GNU* que converteix imatges en format *'pbm',* en arxius de text. L'algoritme de reconeixement està basat en extracció de característiques i té una precisió aproximada de 93% [16].

- *Tesseract* [17]: L'aplicació, ara part de *Google* , llegeix imatges en varis formats i les tradueix a text. És considera un dels OCR més acurat i s'ha portat a varis sistemes operatius. El seu reconeixement, conté aproximadament un 3% d'errors [18].

4. Solució emprada

4.1. Algoritme de segmentació

Després d'analitzar varies alternatives s'ha escollit l'algoritme del treball [8], el toogle-mapping. Les principals característiques de l'algoritme que han fet prendre aquesta decisió són les següents:

- ***Àmbit d'actuació:***
  L'algoritme ha estat dissenyat per a imatges d'escenes reals tant al interior d'edificis com a l'exterior, però a més també tracta qualsevol tipus d'escena controlada, com podrien ser documents escanejats, o qualsevol altre tipus.

- ***Eficiència:***
  Segons el treball aquest algoritme segmenta de manera correcta més caràcters que mètodes com ara l'*ultimate opening* o diversos basats en *thresholding*.

- ***Rapidesa:***
  Ja que es vol executar l'aplicació en un telèfon mòbil es necessita que l'algoritme a utilitzar no tingui molt temps de cpu. El treball afirma que l'algoritme és més ràpid que els mètodes abans descrits.

A més de les característiques principals, aquest mètode és el que ha donat millors resultats amb les proves realitzades sobre els algoritmes abans d'escrits.

4.2. Detecció de text

Com a mètode per a detectar quines de les regions abans segmentades són text i quines no, usarem un filtre previ basat en l'àrea de la regió, és a dir, es filtraran regions que són massa petites o massa grans per a ser text.

Un cop obtinguda la imatge filtrada s'usarà un classificador en forma d'arbre binari, que farà la discriminació de regions que no són text.

En les proves que s'han efectuat, la solució per la que s'ha optat a l'hora de detectar text ha mostrat ser; rapida i senzilla de programar, coses que facilitaran la posterior implementació en *Android*, i eficaç, és a dir, que fa la funció esperada, elimina les regions que no són text amb un encert elevat.

4.3. Reconeixement de text

Un cop analitzades les diferents opcions en OCRs s'ha optat per utilitzar el motor *Tesseract* [17]. Com a punt més important a l'hora d'escollir el motor que detectarà text, s'ha centrat bàsicament en l'eficiència d'aquests. Com s'ha vist en anteriors apartats, el *Tesseract* dona la millor eficiència en quant a reconeixement de caràcters, aproximadament un 97%.

Un altre factor que s'ha tingut en compte és la maduresa d'aquest motor ja que amb això i el seu extens nombre d'usuaris i desenvolupadors, assegurarà que es pot utilitzar amb seguretat en diferents plataformes, i trobar solucions a qualsevol possible entrebanc.

4.4. Visió global del sistema

Un cop escollits tots els components del sistema es pot descriure d'una manera global quin serà el comportament de la nostra aplicació.

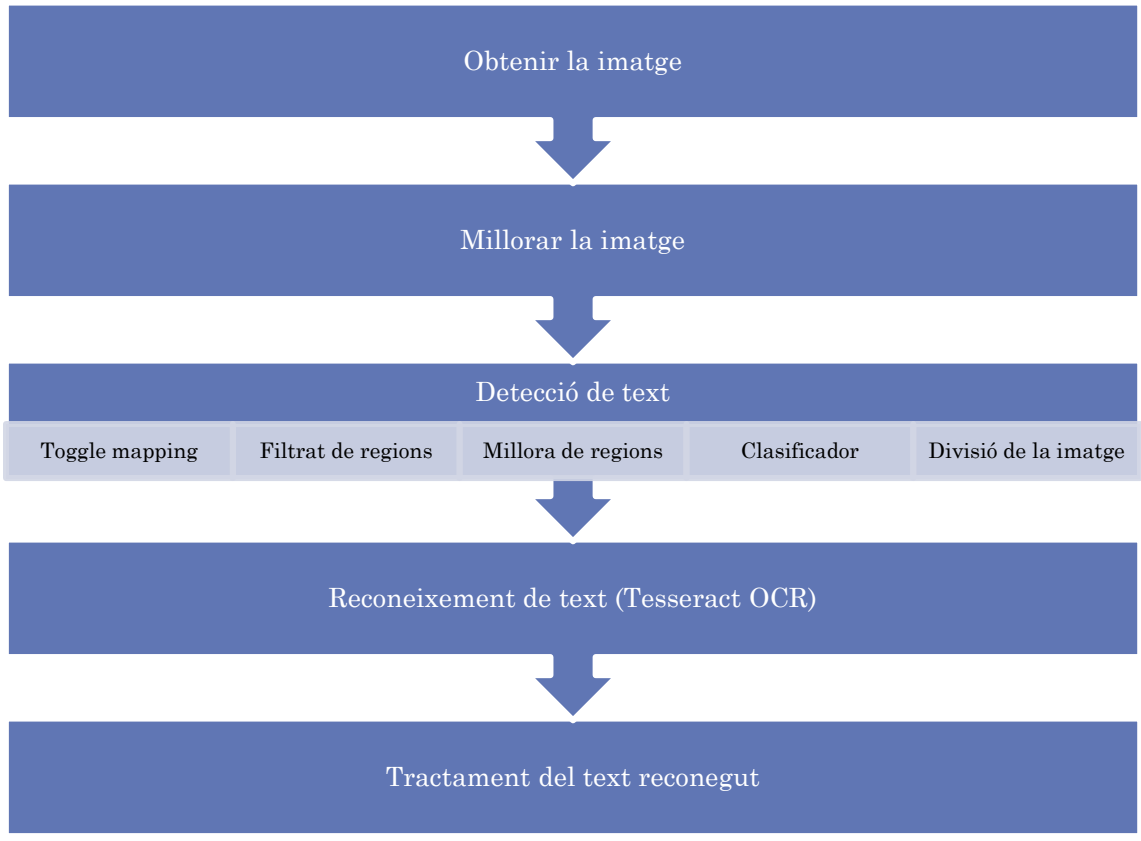


Figura 24 - Visió global del sistema

Com es pot veure en la *figura 24*, la nostra aplicació obtindrà i millorarà una imatge i tot seguit aplicarà el nostre algoritme de *toggle-mapping*.

Tal i com està descrit a la figura amb l'algoritme no serà suficient per a obtenir una imatge correcta per al OCR, per això el nostre algoritme aplicarà un primer filtrat basic sobre característiques locals, un classificador de regions i un divisor d'imatges, del quals es parlarà més endavant. Un cop es té la imatge correctament segmentada i etiquetada l'enviarem al OCR el qual ens retornarà el text reconegut. Un cop obtingut el text, el podrem usar de la manera més convenient per al nostre projecte.

5. Implementació

5.1. Introducció

Per a poder analitzar amb més comoditat els diversos algoritmes i els seus resultats, es divideix el projecte en dos parts:

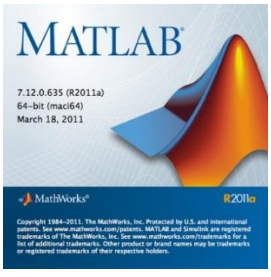
- ***Fase de desenvolupament:***
  En aquesta part es treballa sobre un pc, ja que així s'agilitza la feina de test i validació dels mètodes que es crearan, i no caldrà programar directament sobre el dispositiu mòbil, amb la conseqüent pèrdua de temps. Ja sigui corregint els possibles errors, propis del dispositiu o de comunicació entre el pc i el dispositiu.

- ***Exportació al sistema Android:***
  En el segon tram del projecte, es traduiran els algoritmes ja validats a un llenguatge que pugui interpretar el dispositiu mòbil i es crearà una aplicació que els utilitzi.

5.2. Entorn de treball

Per a la primera part del projecte es treballarà en un ordenador amb sistema operatiu *Mac OS X Lion* en el qual s'usarà el següent software:

- *MATLAB 7.12 (R2011a)* [19]: És un entorn de treball per a computació numèrica, creat per *MathWorks*, que permet manipular i mostrar amb facilitat matrius, funcions i altres tipus de dades, implementar algoritmes i fins i tot comunicar-se amb altres aplicacions i llenguatges de programació. *Matlab* disposa de diferents caixes d'eines (*'toolbox'*) amb les quals es pot expandir el seu àmbit d'actuació. Per a aquest projecte necessitarem la *'Image Processing Toolbox'* la qual permetrà carregar, desar i modificar imatges. S'usarà per al anàlisis i la implementació del mètode de detecció de text així com per a facilitar els tests d'aquest, ja que ens aporta una sèrie de funcions d'alt nivell per a treballar amb facilitat amb imatges.

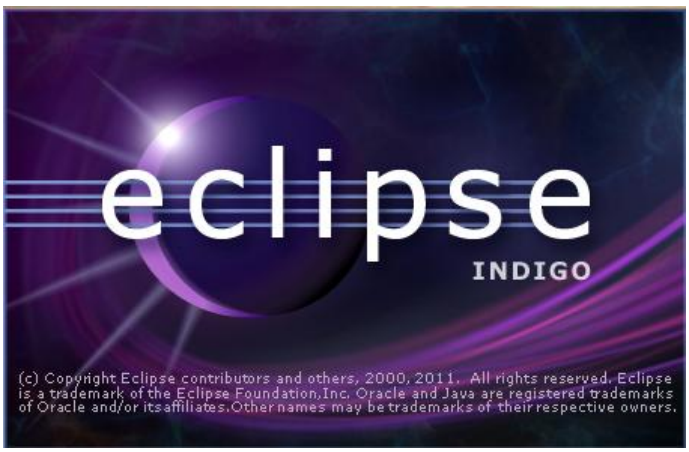


- *TesseractOCR* [20]: És una aplicació per a *Mac OS X* que fa d'interfície d'usuari per al motor OCR *Tesseract*. Permet arrossegar imatges dins el programa i reconèixer el text automàticament. Facilitarà la feina de subministrar les imatges al motor OCR i recollir els resultats d'aquest.

- *Weka 3* [21]: És una aplicació de mineria de dades i d'aprenentatge automàtic, la qual conté un recull d'algorismes per a preprocessat, classificació, regressió, "clustering", associació de regles i visualització de dades. Serà molt útil a l'hora de decidir quins paràmetres usarà el classificador, així com fer un anàlisis de la correctesa dels paràmetres del algoritme.

Per al segon bloc del projecte es desenvoluparà l'aplicació mòbil sobre el mateix ordenador, en el qual es farà ús del següent software:

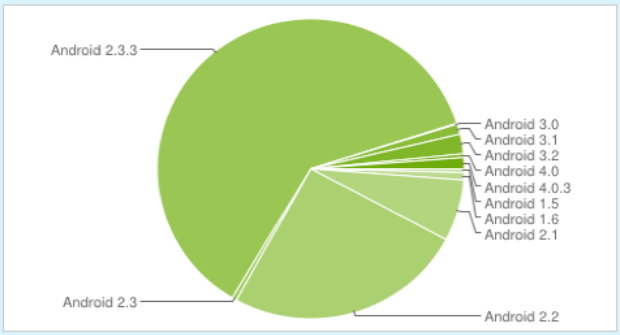
- *Eclipse 3.7.1 (Indigo)* [22]: És un entorn de desenvolupament integrat de codi obert, multi-plataforma, que permet desenvolupar projectes en diferents llenguatges, un cop instal·lats els connectors necessaris, per a cada llenguatge o funcionalitat. Proporcionarà un entorn de treball còmode i ràpid per al desenvolupament del projecte. Per aquestes tasques, es faran servir els següents paquets i connectors:



- *Java Development Kit (JDK)* [23]: Software que proveeix de les eines necessàries per a la creació de programes en *Java*, llenguatge utilitzat per els dispositius *Android*.

- *Android Software Development Kit (SDK) r16* [24]: Software que proveeix de les eines necessàries per a la programació Java d'aplicacions destinades a plataformes amb sistema operatiu *Android*, així com diversos components i APIs per a les diferents versions del sistema. És fàcilment integrable en l'IDE *Eclipse*. En el següent punt es mostra la versió del sistema (i l'API) escollit:

    - *Android 2.2 Platform (API 8, r3)* [25]: S'ha triat aquest sistema com a base ja que, conté totes les funcionalitats necessàries per a desenvolupar la nostra aplicació, així com també per les estadístiques d'us de les diferents versions, que mostren que aquesta plataforma és la segona més usada, just després de la 2.3 que suporta les aplicacions de les anteriors. Com es pot veure en la següent figura, *'Froyo'* amb un 25% es l'opció escollida. [26]



Platform	Codename	API Level	Distribution
Android 1.5	Cupcake	3	0.4%
Android 1.6	Donut	4	0.8%
Android 2.1	Eclair	7	6.6%
Android 2.2	Froyo	8	25.3%
Android 2.3 - Android 2.3.2	Gingerbread	9	0.5%
Android 2.3.3 - Android 2.3.7		10	61.5%
Android 3.0	Honeycomb	11	0.1%
Android 3.1		12	1.1%
Android 3.2		13	2.1%
Android 4.0 - Android 4.0.2	Ice Cream Sandwich	14	0.4%
Android 4.0.3		15	1.2%

Data collected during a 14-day period ending on March 5, 2012

Figura 25 - Versions d'Android segons el seu ús

- *Android Development Tools (ADT) 16.0.1* [27]: Plugin per a *Eclipse* que aporta eines per a desenvolupar ràpidament aplicacions *Android*. Aporta accés des de l'IDE a diverses funcions del SDK així com eines de disseny i construcció d'interfícies d'usuari per a les nostres aplicacions.

- *OpenCV for Android 2.3.1* [28]: És una adaptació del software *OpenCV* per a plataformes *Android*. *OpenCV* és una llibreria de codi obert, que aporta diverses funcions destinades a la visió per computador en temps real. S'utilitzarà per a poder reproduir l'algoritme de detecció de text.

- *Tesseract Android Tools 1.0* [29]: És un conjunt d'APIs per a *Android* que permeten utilitzar el motor OCR *Tesseract*. Per a poder utilitzar el motor *Tesseract* es necessitarà un component *Android* que permeti executar aquest motor en Java.

- *Android Native Development Kit r7 (NDK)* [30]: És un complement per al *Android* SDK que permet tenir parts de la nostra aplicació en codi natiu, ja sigui per rendiment, o per a utilitzar llibreries escrites en llenguatge natiu.

5.3. Fase de desenvolupament del treball

En aquest apartat es detalla com s'ha implementat l' algorisme de detecció i filtrat de text. En la *figura 26* es pot apreciar les etapes de que es composa aquest algorisme, obtenció de la imatge, millora d'aquesta, aplicació del *toggle-mapping* per a la segmentació de la imatge, filtrat de regions que clarament no són text, millora de les regions restants, us d'un classificador per a la selecció de les regions que contenen text, i finalment una divisió de la imatge per a millorar l'eficiència del motor OCR.

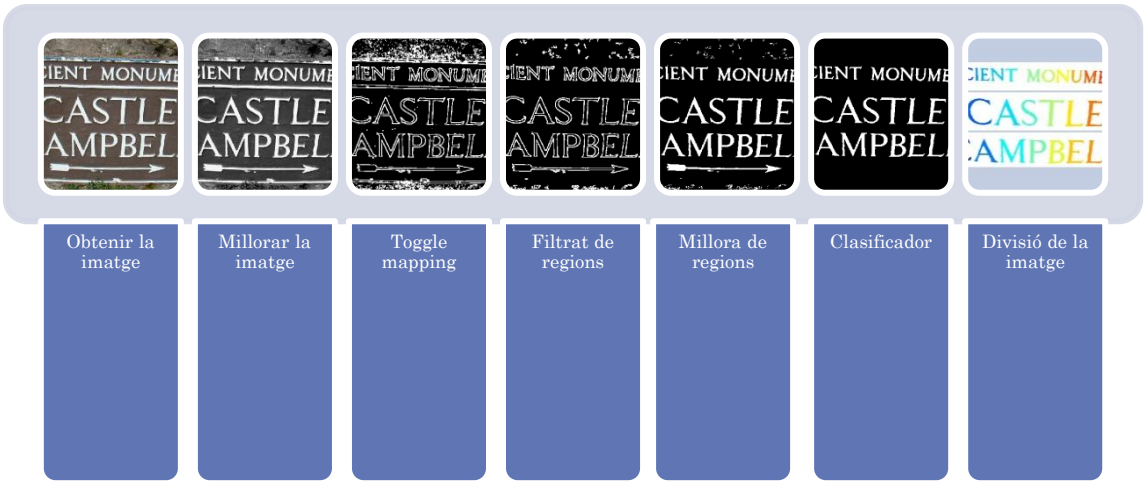


Figura 26 - Visió global del algoritme

5.3.1. Obtenir la imatge

Per a obtenir la imatge original *Matlab* i la *'Image Processing Toolbox'* ofereix una funció anomenada *'imread'* la qual llegeix una imatge en escala de grisos o color de l'arxiu especificat en la cadena de caràcters que passem per paràmetre i la guarda en l'entorn de treball. Amb aquesta funció es pot llegir diversos tipus d'imatges com ara *JPG, GIF, PNG* o *BMP* entre d'altres.

Figura 27 - Imatge obtinguda

5.3.2. Millorar la imatge

Un cop la imatge està carregada s'haurà de transformar a escala de grisos. Això es fa amb la intenció de facilitar els càlculs. L'oportunitat de fer-ho existeix ja que no interessa la informació del color per a reconèixer text sinó que interessa la seva forma.

Per a realitzar aquesta tasca es convertirà la imatge de color veritable a una imatge en escala de grisos. Aquesta funció el que farà serà, convertir les imatges *RGB* a escala de grisos, eliminant el matís i la saturació però mantenint la luminància de cada un dels píxels.

Amb la imatge ja en escala de grisos s'intentarà millorar la imatge usant una funció que mapegi els valors d'intensitat en escala de grisos, converint-los a nous valors. Aquest valors seran tals que l'1% de les dades, està saturat a intensitats baixes i altes de la imatge original. Això implica un augment del contrast de la imatge de sortida, que facilitarà el reconeixement de text.

Figura 28 - Imatge millorada

5.3.3. Toggle-mapping

Un cop obtinguda la imatge preprocessada, es passarà a aplicar-li l'operador de *toggle-mapping* que binaritzarà la imatge. Com ja s'ha vist en apartats anteriors el *toggle-mapping* és un operador morfològic que binaritza una imatge aplicant un operador d'erosió (*h1*) i un altre de dilatació (*h2*) sobre la imatge, per a després aplicar la següent formula que segmenta la imatge.

$$\forall x \in D_f \; s(x) = \begin{cases} 0 & if \; |h_1(x) - h_2(x)| < C_{min} \\ 1 & if \; |h_1(x) - h_2(x)| \geq C_{min} \\ & \& \; |h_1(x) - f(x)| < p * |h_2(x) - f(x)| \\ 2 & altrament \end{cases}$$

Com es pot apreciar en la fórmula, la imatge resultant no es binaria, sinó que conté 3 valors: alt, baix i un tercer, que representa les zones homogènies, és a dir, regions on no hi ha canvi de contrast. Per al problema que s'està resolent no interessa saber quines són les zones homogènies i per tan es modificarà l'algoritme de la forma següent:

$$\forall x \in D_f \; s(x) = \begin{cases} 0 & if \; |h_1(x) - h_2(x)| < C_{min} \\ 1 & altrament \end{cases}$$

Com es pot veure a la fórmula, es té doncs, una variable C_{min} a tenir en compte i es necessitarà l'erosió i la dilatació de la imatge ja millorada.

L'element estructurador que es farà servir per els operadors morfològics, és un quadrat de 3 píxels de costat. S'ha triat aquest, ja que és el que millors resultats ha donat per a imatges en el medi natural, on la vegetació té contrastos similars a les lletres però no forma, ja que les lletres són més uniformes.

Una vegada obtingudes les dos imatges, erosió i dilatació, s'utilitzarà la nova fórmula per a crear una nova imatge binaritzada. El paràmetre de la fórmula C_{min} ha estat ajustat a un valor de 60 després de diversos experiments amb diverses imatges, comprovant quin és el valor òptim de contrast que elimina més parts que no contenen text i no elimina regions amb text. En la *figura 29* es pot veure el resultat d'aplicar aquesta versió del algoritme.

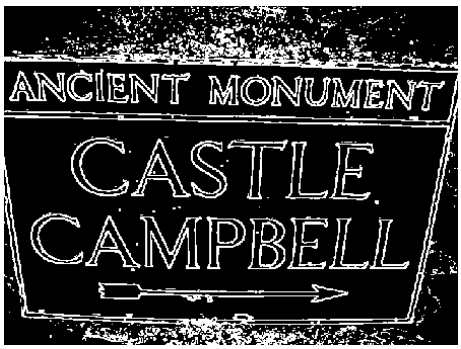


Figura 29 - Imatge binaritzada

Ara que s'ha obtingut la imatge correctament binaritzada, el següent pas és etiquetar-la, és a dir, separar i donar-li nom a cada una de les regions connexes, per a després poder treballar amb elles. Un cop acabat el procés obtindrem una llista de components connexos trobats en la imatge binaria original. Es farà servir una connectivitat per al etiquetatge de 4 píxels.

5.3.4. Filtrat de regions

Ara que ja hem obtingut les regions correctament separades i etiquetades es poden analitzar i així es pot decidir quines són text i quines són soroll.

Aquesta part de l'algoritme és la més costosa en temps de càlcul, ja que per cada component connex de la imatge es calculen diverses característiques d'aquesta, per a poder fer la decisió. Per a intentar reduir aquest temps es farà un filtratge de regions que segur que no són text.

Aquest filtratge, es farà eliminant aquelles regions que són massa petites o massa grans. Això són restriccions que s'han imposat però que es compleixen en la majoria d'imatges. Aquestes restriccions són les llistades a continuació:

- Es calcularà l'àrea com el nombre de píxels de la regió ponderat per la mida de la imatge de la següent manera:
$$area = \#pixels * 1000/area(Imatge)$$
- L'àrea de la regió no serà menor de 0.1
- L'àrea de la regió no serà més gran 40
- L'àrea de la "Bounding Box" de la regió no serà més gran de 120

Un cop eliminades aquestes regions, es tindrà una imatge similar a la mostrada en la part dreta de la *figura 30*.

Figura 30 - Imatge abans i després del filtrat

5.3.5. Millora de regions

Per cada regió resultant del filtratge anterior, se'n millorarà la forma. Com es pot observar a la *figura 30*, només s'han obtingut els contorns de les lletres, ja que és allà on s'hi troba el contrast. Els OCR treballen millor amb lletres solides i no només amb contorns, per tant s'intentarà omplir aquest contorns per a obtenir les lletres desitjades.

Per omplir regions el que es farà és omplir els seus forats. Un forat en una regió, es pot trobar buscant, des de qualsevol dels eixos de la imatge, regions a les que no es pot accedir sense passar per un píxel actiu. En el nostre cas és més senzill ja que les regions abans etiquetades, ja són un contorn tancat i per tant l'únic que s'ha de fer es posar tots els píxels dins la regió com actius. En la *figura 32* es mostra una imatge amb els forats omplerts.

Si el contorn de la lletra que s'ha trobat, no és tancat després de la binarització, com podria ser el cas de la *figura 31*, on la *'O'* ha quedat escapçada, llavors no es podrà omplir l'interior de la regió, i per tant tindrem una lletra en que el OCR tindrà més problemes. Això no preocupa ara en excés, ja que una de les restriccions, és que les lletres no poden ser discontinues en el traç, i per tant no poden tenir un contorn obert, o amb reflexos excessius, cosa que podria fer que al segmentar perdéssim aquesta continuïtat.

Figura 31 - Regió amb contorn obert

Com es pot veure en la següent figura, omplir els forats existents introdueix un altre problema. Caràcters que en principi haurien de tenir forat, com per exemple *'O', 'B'* o *'A',* queden omplerts també, fent-los més difícils de reconèixer.

Figura 32 - Imatge amb forats omplerts

Un altre entrebanc que es troba en la imatge és que l'operador morfològic, el *toggle-mapping*, dilata els contorns, és a dir, deixa els contorns uns píxels més grans del que són en realitat, això no sembla aportar cap problema en caràcters suficientment separats, però com s'observa en la paraula *'ANCIENT'* de la *figura 32*, les lletres *'A'* i *'N'* ara formen part del mateix component connex.

Aquests dos problemes, aportarien una dificultat extra en la feina de diferenciar i reconèixer el text, i per tant, s'intentarà tenir cada un dels caràcters com un sol component connex i amb la forma de la imatge original, o més correctament de la imatge millorada.

Concretament el que es farà és, recuperar la forma original dels caràcters, és a dir, comprovar sobre la regió de la imatge original a tractar, quins són els píxels que realment pertanyen a la regió real, i quins al fons. Així es buidaran els forats de les lletres que ja els contenien i es separaran les lletres que originalment ho estaven, amb un sol mètode. Això s'ha d'aplicar a cada un dels components connexes de la imatge filtrada.

Per a poder comprovar sobre la imatge original si els píxels pertanyen al fons o la regió en qüestió, s'haurà de seguir una sèrie d'etapes mostrades en la *figura 33* i més endavant en la *figura 37*.

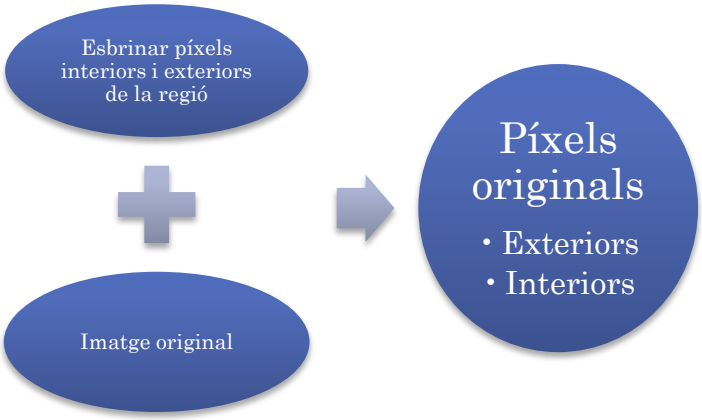


Figura 33 - Etapes de la millora I

Primer s'haurà d'esbrinar si s'està tractant una part amb blanc sobre negre o el contrari, és a dir, si la regió a tractar és més lluminosa o menys que el seu fons, ja que sinó es podrien confondre els caràcters amb el fons i per tant eliminar el caràcter en comptes dels fons.

Per aquest motiu, el que es farà és agafar els píxels interiors a la regió, i calcular la mitja de lluminositat d'aquesta secció, en la imatge original. Si es repeteix el mateix per als píxels exteriors, es podrà comprovar quina de les dues seccions, conté un nivell de llum més alt i quina un de més baix

Per a obtenir les seccions de la regió, es crearan dos mascares. Una que contindrà els píxels interiors i l'altra els exteriors. Aquestes mascares s' obtindran erosionant i dilatant la regió que s'està tractant i després restant, a la regió o de la regió, a tractar, com es mostra en la següent figura, en la qual la imatge de la dreta mostra les dos mascares superposades.

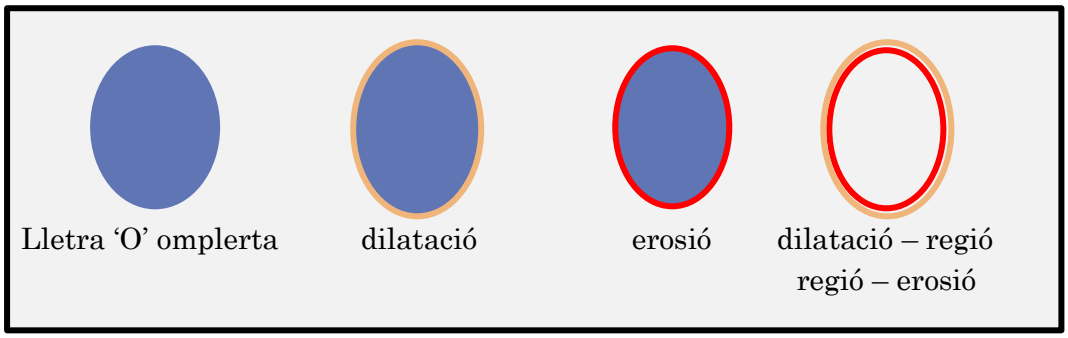


Figura 34 - Màscares de píxels interiors i exteriors a la regió

Com a element estructurant es farà servir un quadrat de 5 píxels de costat. Es podria simplement erosionar/dilatar un sol píxel però per seguretat s'agafarà una franja de 5 píxels.

Un cop les mascares creades, es faran servir per a extreure de la imatge original, els píxels que en cada cas interessen. En la *figura 35* es pot apreciar el resultat d'emmascarar la imatge original sobre el caràcter *'O'* de color negre i el seu fons blanc i en la *figura 36* el mateix caràcter però amb il·luminació oposada. Es fa notar que els píxels amb menys lluminositat no s'aprecien amb claredat, ja que són negres igual que el fons.

Figura 35 - Imatge original, píxels exteriors i píxels interiors (de l'original) I

Figura 36 - Imatge original, píxels exteriors i píxels interiors (de l'original) II

Amb la informació obtinguda ara, es podrà seguir les etapes mostrades en la *figura 37* fins a recuperar la forma original del caràcter.

Figura 37 - Etapes de la millora II

Un cop obtinguts els píxels originals, es pot comprovar quina de les seccions conté una lluminositat més baixa, i quina més alta, i llavors decidir si s'està tractant blanc sobre negre o viceversa. Amb aquesta informació, ja es podrà recuperar els píxels que s'haguessin pogut perdre en el procés de binarització, i per tant, tenir de nou la forma original.

Per a recuperar els píxels, de la imatge original millorada, s'agafarà la regió que s'està tractant i es binaritzarà amb un llindar global fix. Amb aquest procés podria semblar que es perd tota la feina realitzada per a obtenir els contorns, però no és així, ja que la regió que s'està tractant, es suficientment petita per poder aplicar aquest mètode de binarització sense errors. Per tal de trobar aquest llindar global, s'utilitzarà el mètode de *Otsu*, el qual agafa el nivell que minimitza la variància intraclasse de píxels blancs i negres.

En la *figura 38* es pot veure el que seria el caràcter *'O'* en la imatge original però, binaritzada. A la dreta podem veure el caràcter sobre fons blanc i a l'esquerra sobre fons negre.

Figura 38 - Caràcters originals binaritzats

Amb la regió binaritzada i la informació sobre la lluminositat de les seccions, es podrà recuperar la forma original seguint les regles següents. Si la regió que s'està tractant es blanc sobre negre (píxels exteriors amb lluminositat superior), l'únic que s'ha de fer, és eliminar de la regió, aquells píxels que no estiguin actius en l'original binaritzada, i afegir aquells, que estiguin actius en l'original. En cas contrari, negre sobre blanc, primer s'haurà de fer el negatiu de la imatge original, per a després, eliminar i afegir píxels tal com hem explicat.

Un cop analitzat cada component connex es podrà crear una imatge final amb la intersecció de totes les imatges parcials de cada component, per a obtenir la imatge amb les regions millorades com la de la dreta de la *figura 39*. Es pot comparar aquesta amb la part esquerra de la figura i comprovar com els caràcters són molt més fàcils de reconèixer després d'aquest algoritme.

Figura 39 - Millora de les regions

5.3.6. Classificador

En aquets punt es té una imatge ja millorada, binaritzada, i amb un primer filtre com la de la dreta de la *figura 39*, però com es pot apreciar encara queda soroll en la imatge, que podria ser confós amb un caràcter per part del OCR. En conseqüència, s'haurà de crear un filtre, que separi el que són caràcters del soroll de fons.

Això s'aconseguirà analitzant les propietats de cada regió connexa. Algunes de les propietats que podrien ajudar a diferenciar els caràcters del soroll són les següents:

- *'Àrea':* Escalar que especifica el nombre de píxels de la regió.
- *'BoundingBox':* És el rectangle més petit que conté tota la regió.
- *'Excentricitat':* Escalar que especifica la distancia entre, el focus de l'el·lipse que té el mateix segon moment que la regió i l'eix major de la mateixa el·lipse.
- *'Nombre d'Euler':* Escalar que especifica el nombre d'objectes de la regió menys el nombre de forats d'aquests objectes.
- *'Extensió':* Escalar que especifica el ràtio entre el nombre de píxels de la regió i el nombre de píxels de la *'BoundingBox'*.
- *'Àrea omplerta':* Escalar que especifica el nombre de píxels de la regió però, amb tots els píxels actius.
- *'Eix menor':* Escalar que especifica la distancia del eix menor de l'el·lipse amb el mateix segon moment que la regió.
- *'Eix major':* Escalar que especifica la distancia del eix major de l'el·lipse amb el mateix segon moment que la regió.
- *'Orientació':* Escalar que especifica l'angle, en graus, entre l'eix x i el eix major de l'el·lipse que té el mateix segon moment que la regió.
- *'Perímetre':* Escalar que especifica la distancia en el contorn exterior de la regió.
- *'Solidesa':* Escalar que especifica la proporció de píxels en el menor polígon convex que també són a la regió.
- *'Aspect':* Escalar que especifica el ràtio entre el *'Eix major'* i el *'Eix menor'* de la regió.
- *'Compacitat':* Escalar sense unitat que ens especifica l'àrea de la regió dividida per el perímetre al quadrat d'aquesta.

A partir d'algunes d'aquestes propietats anteriors es podria parametritzar el que és un caràcter i el que no. Per a decidir quines són les propietats que millor descriuen els caràcters i per tant els diferencien més del soroll, es farà servir una eina de mineria de dades anomenada *Weka*.

Amb aquesta aplicació es farà un anàlisis de les diferents propietats que ja s'han llistat i es mostrarà una llista amb aquelles que són més influents, a l'hora de decidir si el component que s'està tractant és un caràcter. S'usarà com a classificador el "CFS Feature Set Evaluation" el qual dona puntuacions a subgrups d'atributs per a després escollir el millor subgrup. Per a això es necessitaran varies imatges d'entrenament, en les quals, s'etiquetarà cada regió, amb totes les seves propietats, i un booleà que indicarà si és un caràcter o no.

Per a fer l'entrenament s'han triat 13 imatges, que són una bona representació de tots els tipus d'imatge possibles, com per exemple, lletres més grans o més petites, més resolució o menys, escena interior o exterior, lletra clara sobre fons clar o viceversa, etcètera.

Figura 40 - Imatges d'entrenament

En la figura anterior, es mostren les imatges que es faran servir per entrenar el classificador. Com es pot veure l'ultima imatge (*'Marlboro lights'*) té dos caràcters escapçats per un reflex, que descartarem com a soroll, ja que no podem veure tot el caràcter, hem escollit aquesta imatge "defectuosa", perquè ens interessa la tipografia mostrada.

Per tant, aquestes imatges seran les imatges tipus, que l'algoritme serà capaç de reconèixer amb més eficàcia.

S'ha de tenir en compte que, com abans, l'àrea i el perímetre han estat ponderats per la mida de la imatge. Després d'analitzar els atributs amb el set d'entrenament anterior, s'ha descobert que els paràmetres que prediuen amb claredat si la regió és un caràcter són els següents:

- ✓ *'Area'*
- ✓ *'Aspect'*
- ✓ *'Compacity'*
- ✓ *'Extent'*

Ara que es coneixen quins són els paràmetres que més influeixen sobre la decisió, es pot crear un classificador amb només les propietats prèviament elegides.

Per simplicitat d'implementació i per els resultats obtinguts amb ell, s'ha triat fer un arbre de decisió per a construir el nostre classificador. Per a crear l'arbre, es farà a partir d'arbres aleatoris validats contra el propi test d'entrenament. S'ha escollit aquesta opció pels resultats, ja que ens genera la *'confusion matrix'* de la *taula 3*, la qual es totalment diagonal i simètrica.

Classificat com ha	Lletra	Soroll
Lletra	197	0
Soroll	0	399

Taula 3 - "Confusion matrix" del classificador

En la següent figura es pot veure l'arbre de decisió que s'ha generat basat en les imatges d'entrenament que s'han utilitzat. Es fa notar que s'han esborrat totes les fulles del arbre que marcaven una regió com a lletra, ja que no interessaran, només es vol saber les que s'han d'eliminar.

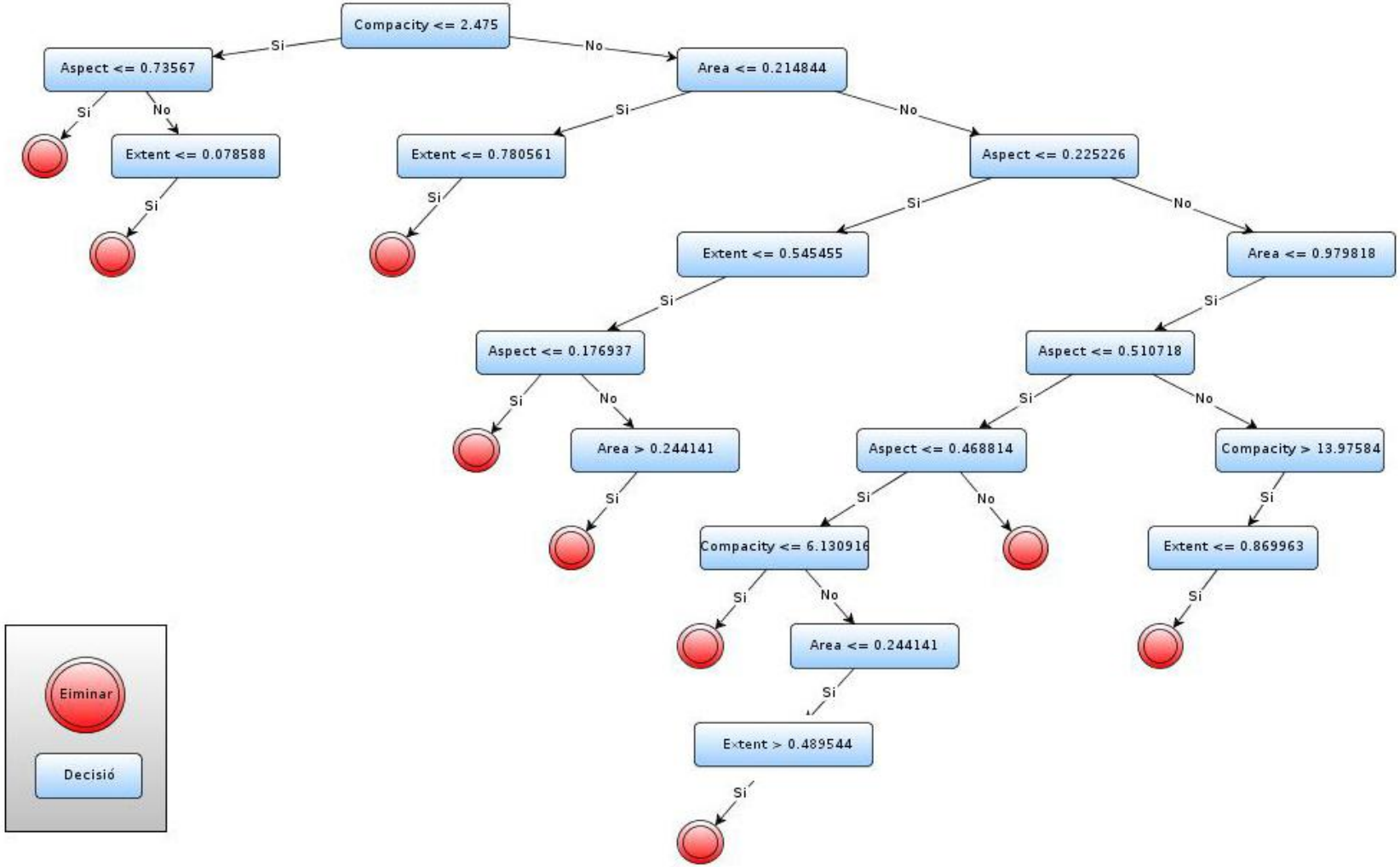


Figura 41 - Arbre de decisió

Un cop obtingut l'arbre de decisió, es pot especificar per a cada una de les regions si és una lletra o no. Totes aquelles que no compleixin les restriccions del arbre seran eliminades. Cal tenir en compte que abans d'usar l'arbre de decisió, s'ha tornat a fer un filtratge senzill sobre les regions molt similar al explicat anteriorment, però amb els següent paràmetres:

- Es calcula l'àrea com el nombre de píxels de la regió ponderat per la mida de la imatge de la següent manera:
$$area = \#pixels * 1000/area(Imatge)$$
- L'àrea de la regió no serà menor de 0.1
- L'àrea de la regió no serà més gran 80
- L'àrea de la *'Bounding box'* de la regió no serà més gran de 120

Com es pot apreciar en la següent figura la imatge a quedat correctament binaritzada, segmentada i amb només regions que contenen text.

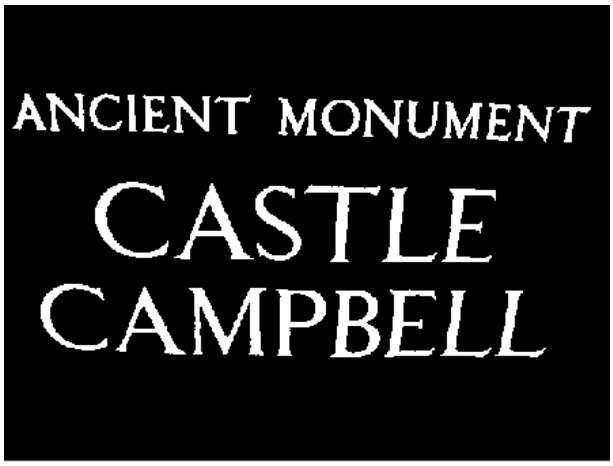


Figura 42 - Imatge filtrada pel classificador

5.3.7. Divisió de la imatge

Un cop obtinguda la imatge correctament binaritzada i filtrada de totes o gairebé totes les regions que no contenen text, es podria enviar aquesta imatge directament al motor OCR. Tot i això, facilitarem la feina al motor *Tesseract* per a que no treballi amb imatges d'entrada on hi puguin haver caràcters de tipologia o mida, molt diferenciades entre elles, com podria ser la diferencia de mida entre la primera línia i les següents de la *figura 42*.

Ja que les tipologies i mides de caràcters diferents acostumen a ser en línies diferents de la imatge el que es farà és, dividir, si és possible, la imatge horitzontalment en cada una de les diverses línies o frases que conté la imatge.

Per aconseguir les diferents regions es buscarà files de píxels horitzontals que només continguin fons, és a dir, cap regió, i llavors es podrà dividir la imatge per aquestes files.

Per a buscar files horitzontals sense regions, el que es farà és fer la projecció de la imatge sobre l'eix *'y'* (alçada). La projecció sobre l'eix *'y'*, es calcularà, amb l'acumulació dels píxels per a cada una de les files de la imatge. Amb aquesta informació, s'ha de buscar aquelles files que continguin un zero, dividir la imatge per aquetes files i eliminar aquelles que continguin zeros.

Si això es fa de forma seqüencial, es trobaran canvis de zeros a valors positius i de valors positius a zeros. Si en cada 2 d'aquests canvis, es crea una nova imatge amb els píxels entre aquest dos canvis, s'obtindrà la imatge subdividida, sense escapçar cap dels caràcters.

En la següent figura es pot veure les imatges subdividies de la *figura 42*, a més s'ha pintat cada component connex d'un color diferent per apreciar-los millor.

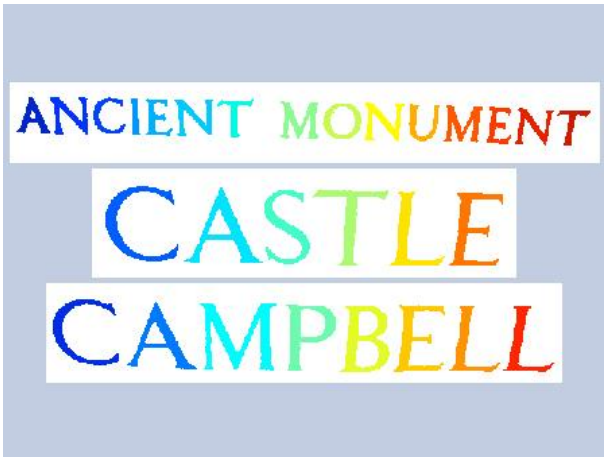


Figura 43 - Imatge dividida i pintada segons components connexos

La *figura 44* té un resum de les tècniques de processat que s'han aplicat sobre la imatge original:

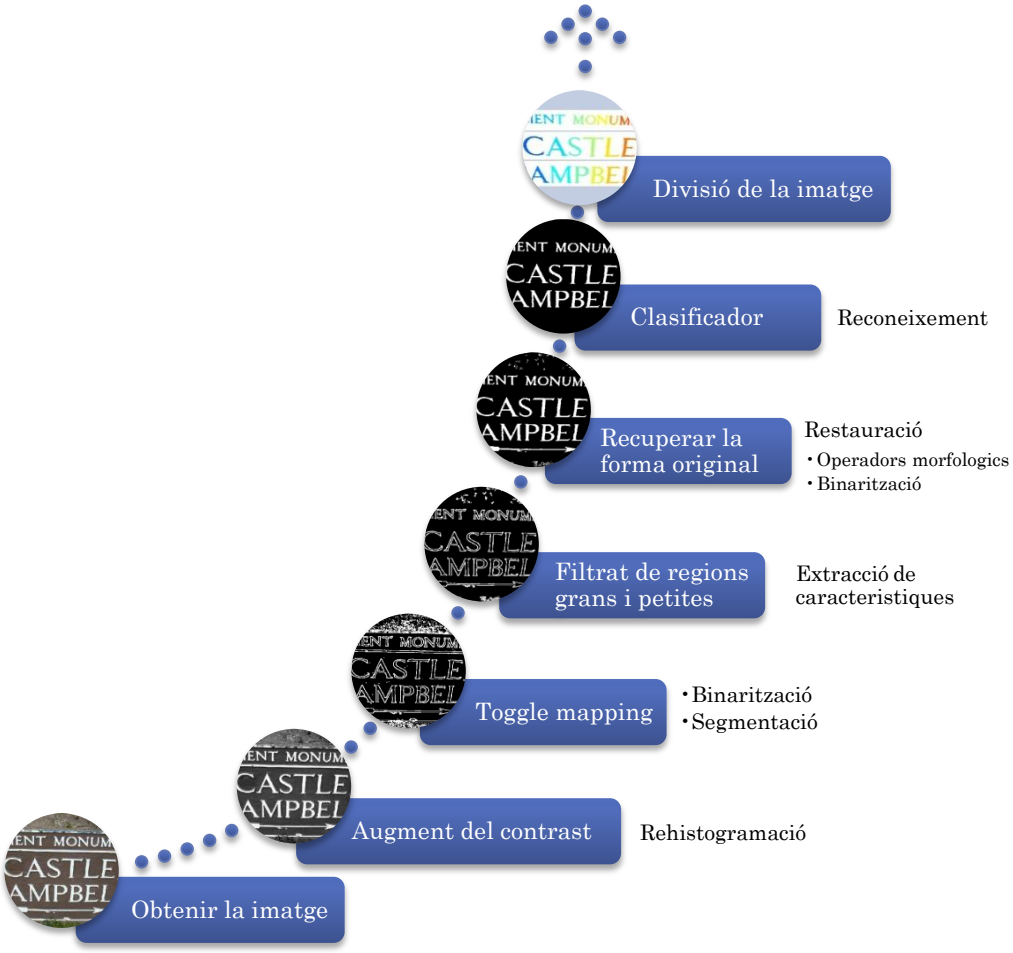


Figura 44 - Etapes del processat de la imatge

5.3.8. Text reconegut

Un cop es té la imatge llesta, es hora d'utilitzar l'OCR. Com es pot veure en les *figures 43* i *44*, per a utilitzar l'aplicació s'ha d'arrossegar la imatge desitjada al requadre esquerra i automàticament fa la crida al motor i retorna en el quadre de la dreta el text resultat del reconeixement.

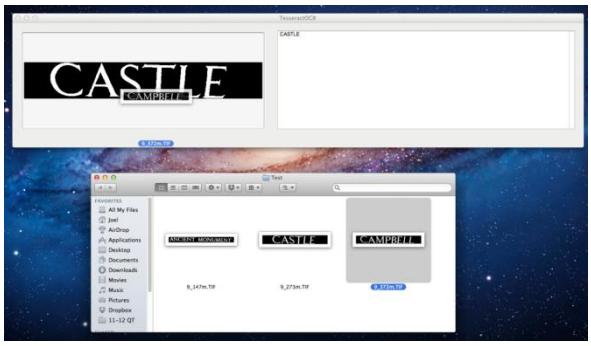


Figura 45 - Ús del Tesseract OCR

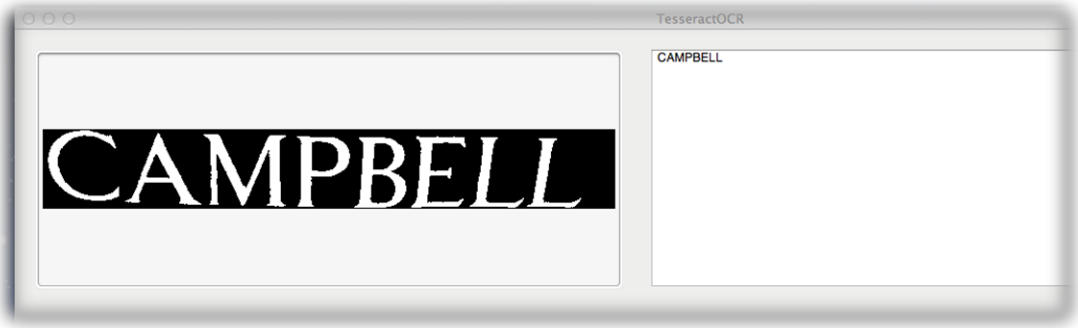


Figura 46 - Resultat del Tesseract OCR

5.4. Desenvolupament de l'aplicació en Android

Figura 47 - Icona de DroidText

En aquest apartat es mostra el desenvolupament de l'aplicació *Android,* la qual s'ha anomenat *DroidText*. L'aplicació es provarà sobre un telèfon *HTC Desire*, amb, entre d'altres, les característiques mostrades en la següent taula.

Característica	*Detall*
CPU	1 GHz Qualcomm QSD8250
GPU	Adreno 200 (AMD Z430)
Memòria	576 MB RAM
Memòria externa	2 GB micro SDHC (fins a 32 GB)
Pantalla tàctil	3.7-inch 480×800 WVGA AMOLED
Càmera	5 Megapíxel, Autofocus, LED flash, Zoom
Sistema operatiu	Android 2.2 Froyo

Taula 4 - Característiques HTC Desire

L'aplicació obtindrà una imatge de la càmera incorporada i la tractarà amb l'algoritme que acabem de dissenyar. Un cop es té una imatge tractada, l'aplicació serà capaç d'executar l'OCR *Tesseract* per a obtenir el text en format digital present en la imatge tractada.

5.4.1. Estructura de l'aplicació

Com ja s'ha explicat, l'aplicació serà programada en *Java*, això significa, que es tindrà una sèrie de classes *Java* que interactuaran entre elles. L'estructura bàsica de l'aplicació es mostra en la següent figura.

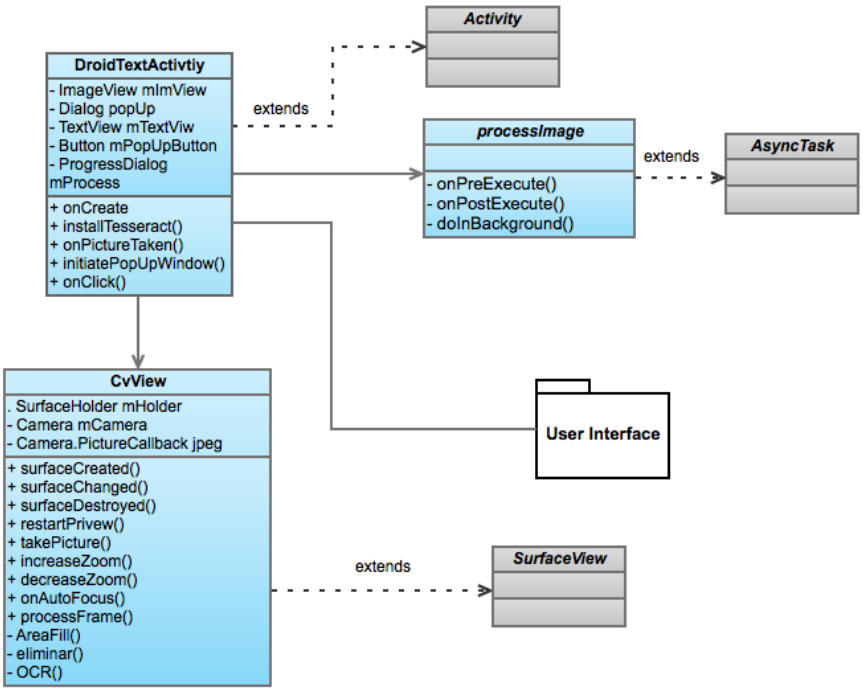


Figura 48 - Diagrama de classes de l'aplicació

Com es pot apreciar, l'aplicació tindrà una sola activitat (*DroidTextActivity*), amb la seva interfície d'usuari, de la qual es parla en l'*apartat 5.4.5*.

Aquesta activitat, s'encarregarà de crear les classes *CvView* i *processImage* les quals duran a terme l'obtenció, el tractament de la imatge i, un cop obtinguda la imatge ja dividida en diverses línies, s'obtindrà el text reconegut en format digital.

En la següent figura es pot apreciar la seqüencia bàsica de l'aplicació.

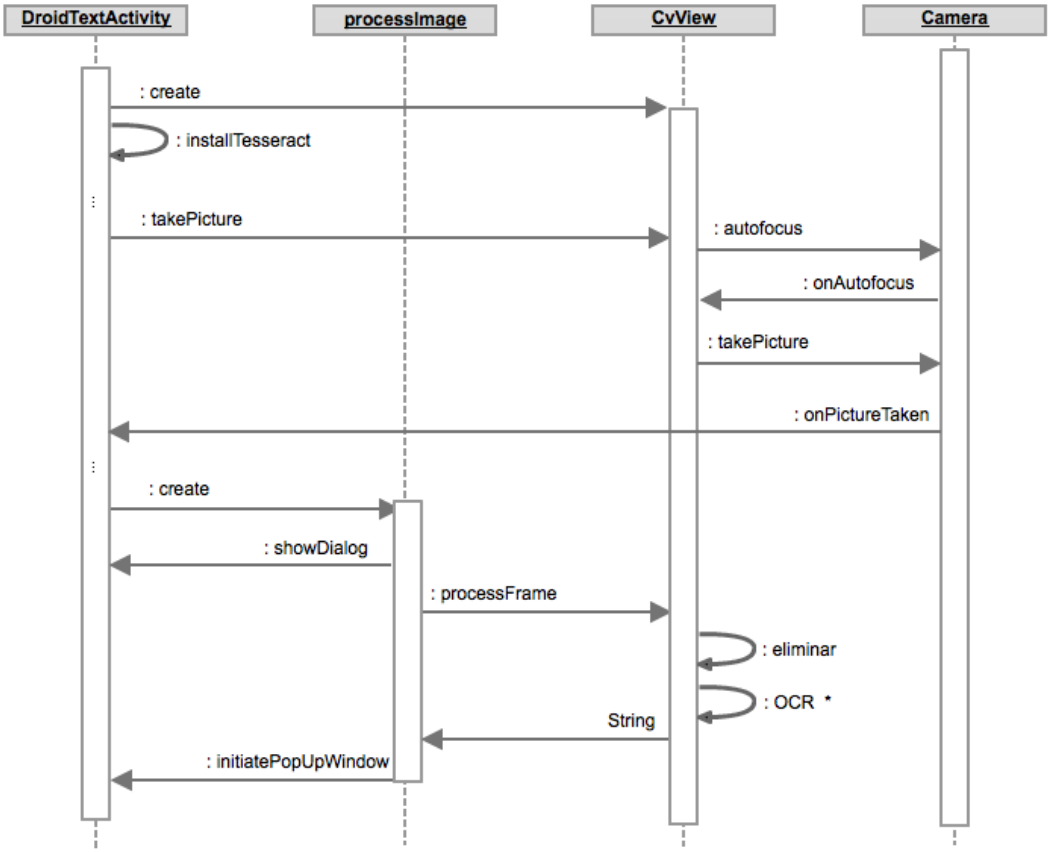


Figura 49 - Diagrama de seqüencia de l'aplicació

Al iniciar l'aplicació, s'inicialitzen totes les interfícies gràfiques, de les quals es parla més endavant, i es crea la classe *CvView*, que farà de gestor de la càmera. Aquesta classe conté un *SurfaceHolder*, en el qual es pot configurar la càmera perquè mostri una vista prèvia del que la càmera esta obtenint.

Un cop configurada la càmera, s'instal·larà el *OCR Tesseract*. Aquesta funció comprova si tots els components que necessita el OCR són presents en la memòria externa. Aquest components són els diversos components de cada llenguatge.

Per exemple, si es vol reconèixer text en anglès, s'haurà de tenir guardat en la memòria externa el fitxer, *"eng.traineddata"*, el qual conté les dades necessàries per a reconèixer text en aquest idioma.

Si no existeixen aquests components en la memòria externa, es crearà una nova carpeta en l'arrel de la memòria SD, amb el nom de la nostra aplicació (*/DroidText*), i dins s'hi copiaran els components necessaris.

Un cop correctament configurada l'aplicació, i amb la vista prèvia de la càmera funcionant, és el moment d'obtenir la imatge de la qual es vol reconèixer text.

5.4.2. Obtenir la imatge

Quant l'usuari està llest per a obtenir la imatge desitjada, l'activitat llença la funció *takePicture* de la classe *CvView*. Aquesta funció és l'encarregada de que, un cop ja configurats tots els paràmetres de la càmera, obtenir la imatge, el més nítida i enfocada possible, per després retornar-la a l'activitat.

Com ja s'ha explicat, al crear la classe *CvView*, s'haurà de linkar el *SurfaceHolder* amb la vista prèvia de la càmera, així com establir els paràmetres de la càmera com poden ser, la resolució, la lluminositat o la disponibilitat de flash o zoom.

Concretament l'aplicació obtindrà totes les possibles resolucions d'imatges que la càmera pot retornar, i seleccionarà aquella que més s'adeqüi a la pantalla del dispositiu mòbil que s'està usant. Com que es pot tenir diferents mides i resolucions de pantalla, l'algoritme intentarà ajustar la imatge de forma que la pantalla usada i la imatge tinguin la resolució més semblant possible.

Un cop configurada la resolució, es configurarà la càmera per a que s'adapti automàticament a la lluminositat de l'ambient, activarem el flash automàtic i s'iniciarà la vista prèvia d'imatges.

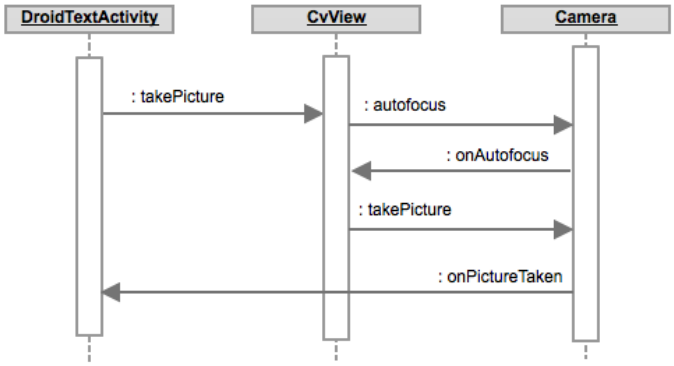


Figura 50 - Diagrama de seqüencia | Obtenir imatge

Per a obtenir la imatge, es necessiten dos passos bàsics, com es mostra en la figura anterior, l'autofocus i la obtenció de la imatge. L'autofocus és una funció tal que al cridar-la, la càmera intenta enfocar la imatge de la vista prèvia, i un cop ha acabat d'enfocar, retorna un senyal a la classe desitjada.

Un cop la càmera està enfocada en el motiu que es vol capturar, l'obtenció de la imatge, es duu a terme amb la funció *takePicture* de la classe *Camera*. Aquesta funció, retornarà el senyal de finalitzat a la classe desitjada juntament amb la imatge obtinguda codificada com un vector de bytes, els quals representen la imatge.

En el següent apartat es mostra com descodificar la imatge i processar-la amb l'algoritme que s'ha dissenyat, i obtenir una imatge amb el mínim residu possible de regions, que no són text.

5.4.3. Processament de la imatge

En executar-se la funció *onPictureTaken*, la imatge ja ha estat capturada, i es retornada, com be ja s'ha explicat, en forma d'un vector de bytes. Un cop l'activitat a rebut la imatge, a part de diverses operacions orientades a la interfície, les quals s'expliquen més endavant, es llença la funció *processFrame* de la classe *CvView*, la qual s'encarrega de la feina de processar la imatge amb l'algoritme ja dissenyat, executar l'OCR, i retornar el text reconegut.

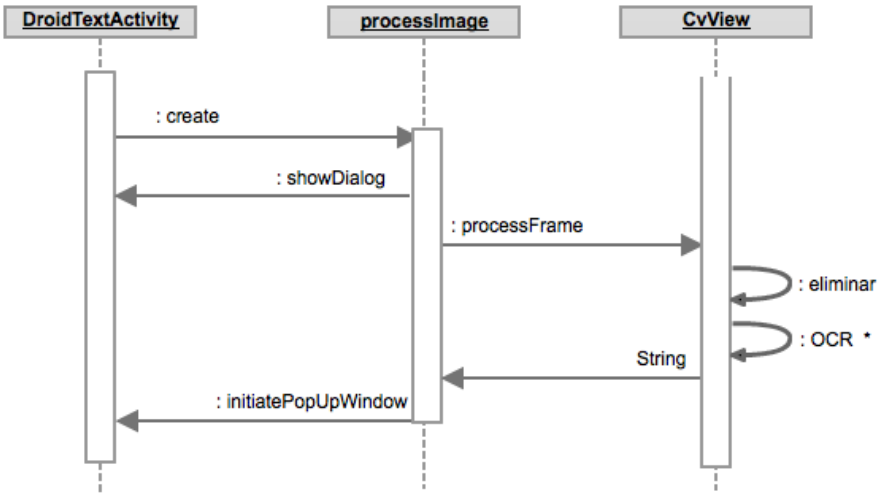


Figura 51 - Diagrama de seqüencia | Processat

En l'anterior imatge es pot observar un diagrama de seqüencia bàsic de les operacions utilitzades. La classe *processImage* és una classe que es pot executar asíncronament, i que facilita gestions de la interfície d'usuari com es veurà més endavant.

Per a processar la imatge, es farà servir l'algoritme dissenyat en la *secció 5.3*, del qual s'haurà de traslladar les funcions i operacions creades en *MATLAB*, a *Java*. A això hi ajudarà *OpenCV*, una llibreria designada per a les tasques de visió per computador, amb totes les operacions bàsiques que es necessiten, com poden ser, l'erosió , dilatació o intersecció d'imatges, així com la segmentació d'aquestes.

Abans de començar amb el procés en sí, s'haurà de transformar el vector de bytes que rep la funció, el que s'ha obtingut de la càmera, i transformar-lo en una matriu de píxels en la qual puguem treballar.

El vector de bytes rebuts està codificat amb el sistema de color YUV 4:2:0, el qual té un valor Y (luminància) per cada píxel de la imatge, i un valor U i un altre V (crominàncies) per cada cel·la 2x2 de la imatge, com es mostra en la següent figura.

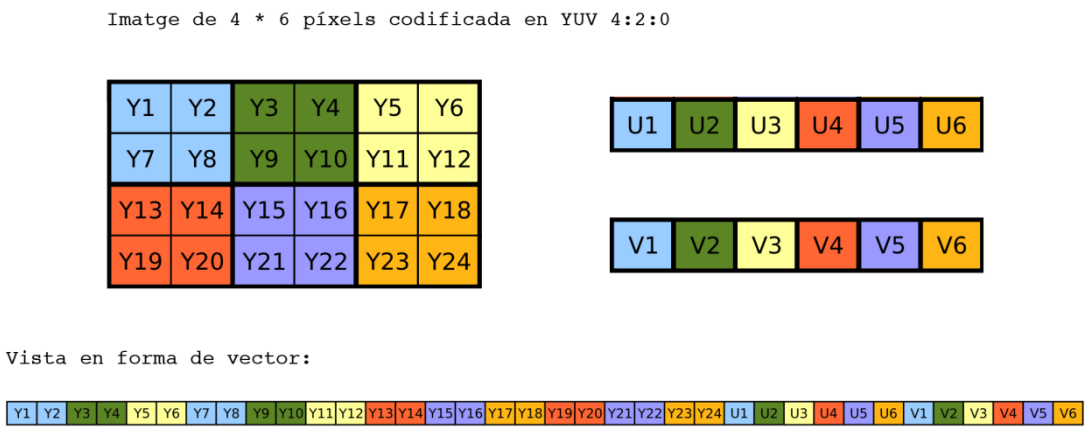


Figura 52 - Format YUV 4:2:0

Gracies a la informació Y del vector, s'obtindrà una matriu amb la imatge ja en escala de grisos. A partir d'aquí, es començarà el procés del algoritme, el qual està detallat a la *secció 5.3*. Primer amb l'aplicació de la nostra versió del "toggle-mapping". Tot seguit, s'etiquetarà cada regió, i s'aplicarà el filtratge ràpid. Un cop els CC han estat filtrats, s'aplicarà la millora de regions proposada, per a després classificar les regions amb l'arbre de decisió creat.

A partir d'aquest punt, l'algoritme proposat difereix de la versió original teòrica, en el càlcul d'una de les característiques que són analitzades per l'arbre de decisió. Aquesta característica és l' *'Aspect'*, el qual era calculat com el ràtio entre els eixos de l'el·lipse que té el mateix segon moment que la regió.

Aquesta el·lipse és molt costosa a l'hora de calcular, i per tant augmentaria molt el temps de càlcul de l'aplicació.

Per això, s'ha decidit substituir la manera de calcular la característica, essent ara aquesta, el ràtio entre els eixos de la *'Bounding Box'* de la regió. Com es pot veure en la part esquerra de la següent figura, en la majoria de regions, la diferencia entre els mètodes de calcular aquesta característica és mínima.

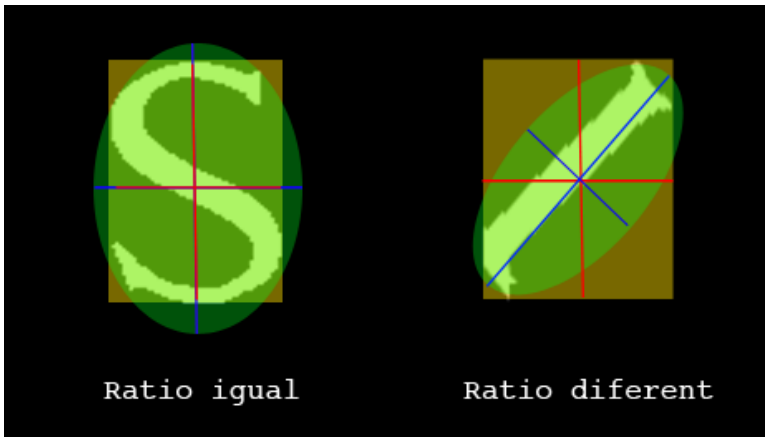


Figura 53 - El·lipse vs. "Bounding Box"

En la resta de regions, com pot ser la regió de la dreta de la figura anterior, si aquesta havia de ser eliminada, serà eliminada, amb alta probabilitat, per altres característiques. Les poques regions restants, seran acceptades, el que produirà l'addició d'algun fals positiu. Aquest fals positiu, és possible que sigui confós per un caràcter per l'OCR, però és un risc necessari per al funcionament fluid de l'aplicació.

Un cop eliminades les regions segons l'arbre de decisió, es divideix la imatge en files, com ja s'ha explicat anteriorment. Cada una de les imatges generades serà enviada al OCR, per al reconeixement del text.

5.4.4. Execució del OCR

Ara que tenim cada fila de la imatge, haurem de generar un *Bitmap Java* a partir de la matriu que s'ha creat. Un *Bitmap* és una forma de representar la imatge, i és la triada per *Teseract* com a entrada del seu algoritme. El següent esquema mostra el funcionament de l'execució del OCR.

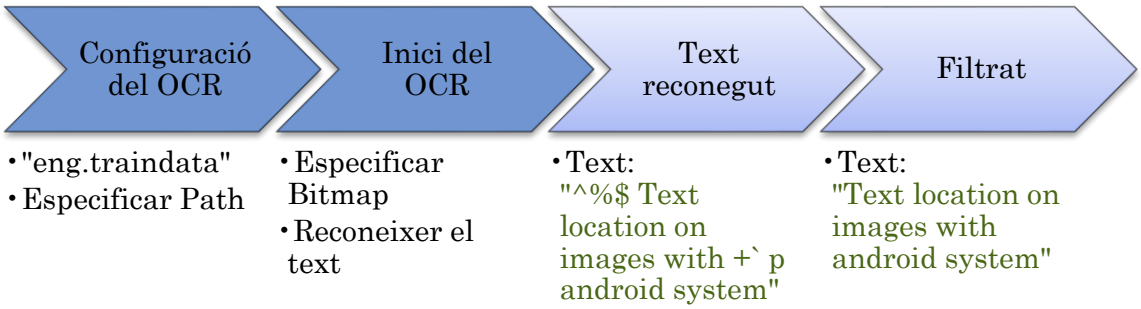


Figura 54 - Procés del OCR Tesseract en Android

Primer s'ha de crear i configurar el Motor OCR, és a dir, crear la classe responsable de la interacció amb l'OCR, i configurar aquesta amb el component de llenguatge desitjat. En aquest cas, s'usarà com a idioma per reconèixer, l'anglès, per tant haurem d'especificar al *Tesseract* on es troba el component corresponent. En qualsevol moment aquest idioma podrà ser canviat entre un dels possibles.

Un cop ja configurat l'OCR, s'haurà de dir-li quina és la imatge que volem tractar, i demanar que retorni el text present en aquesta imatge. El text reconegut pot contenir caràcters com poden ser *'/', '{'* o *'*'*. Aquests, acostumen a ser reconeguts de regions que eren falsos positius, o d'errors de reconeixement del propi OCR. És per això que al text reconegut se li passarà un filtre de caràcters. Aquest filtre només acceptarà, les lletres del abecedari (*"A..Za..z"*), els dígits (*"0..9"*), i algun símbol específic que pugui ser important de reconèixer.

A més també s'eliminaran tots els espais inútils, així com lletres soltes que hagin pogut ser reconegudes, excepte és clar les vocals i la i grega, ja que són les úniques lletres que poden anar separades d'una paraula.

Un cop es té cada línia de text reconeguda, es crearà un *String* amb totes elles, i es retornarà a l'activitat principal, per a que pugui mostrar el text reconegut en format digital.

5.4.5. Interfície gràfica

En aquest apartat es veurà la seqüencia d'operacions que fa l'aplicació des del punt de vista de l'usuari mostrant en cada cas, que fa l'usuari i que se li mostra a aquest.

Un cop instal·lada, s'obrirà l'aplicació tal i com es mostra en la següent figura.

Figura 55 - Obrir l'aplicació

La primera cosa que farà l'aplicació en carregar-se serà, com ja s'ha explicat, instal·lar el *Tesseract,* si és necessari, i preparar la vista prèvia de la càmera, per a mostrar-la a l'usuari. La següent figura mostra la situació inicial de l'aplicació.

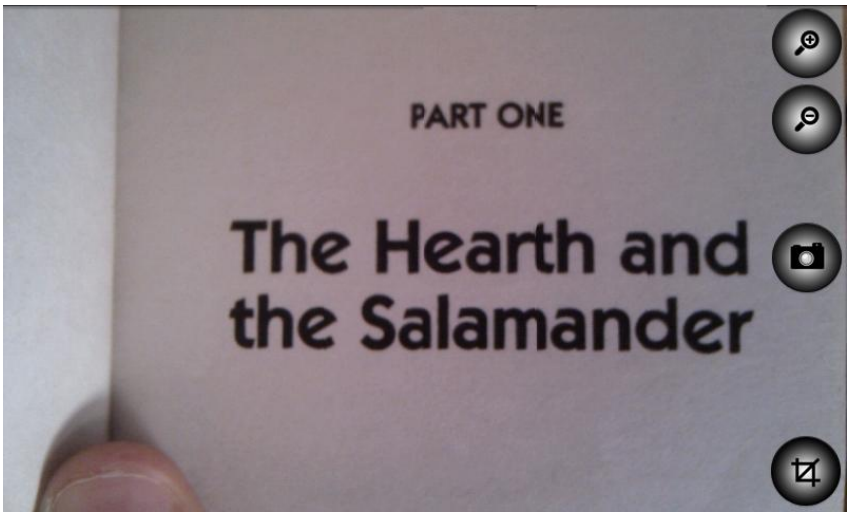


Figura 56 - UI - Pantalla inicial

Com es pot apreciar, tota la pantalla mostra una previsualització del que la càmera observa en cada moment. A més la interfície consta de quatre botons, els quals no són botons físics del telèfon mòbil, sinó que són inclosos en la vista de la interfície, i poden ser activats prement sobre ells tàctilment.

Tot i això els botons poden obtenir el focus de l'aplicació, seleccionant-los amb els botons físics del dispositiu mòbil (dreta, esquerra, amunt i avall) i poden ser clicats amb la tecla d'acció del mateix dispositiu, per aquells dispositius mòbils que no disposin de pantalla tàctil, o usuaris que no vulguin usar-la en aquest moment.

La interfície disposa de dos botons de zoom, situats a la part superior dreta de la interfície. Aquests, augmenten o disminueixen la distancia amb l'escena a capturar. Al ser premuts, envien una notificació a l'activitat, que en ser escoltada, executa la funció de zoom desitjada sobre la classe *CvView*, la qual cridarà a la càmera per a fer un pas de zoom en la direcció indicada.

Aquestes funcions, demanen a la càmera si posseeix l'habilitat de fer zoom, i si és així, apliquen el zoom demanat, en una unitat. Cal tenir en compte que cada dispositiu té la seva càmera, i cada càmera pot tenir prestacions molt diferents. En la següent figura es mostra la mateixa imatge que la *figura 56*, amb l'ampliació zoom de la vista de la càmera.

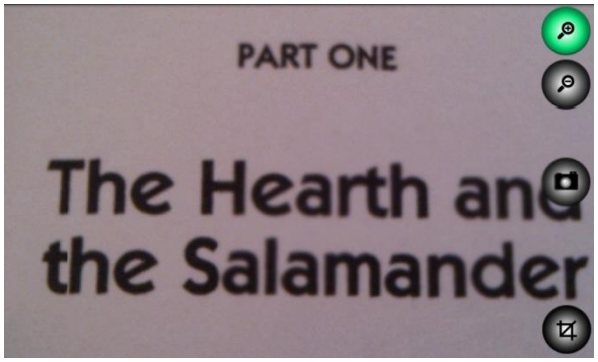


Figura 57 - UI - Utilització del zoom

Un cop es té l'escena que volem capturar en la previsualització, l'usuari pot prémer el botó de captura, situat al centre del marge esquerra de la interfície. Aquest, desencadenarà el procés de l'activitat per a capturar la imatge, processar-la i retornar-ne el resultat.

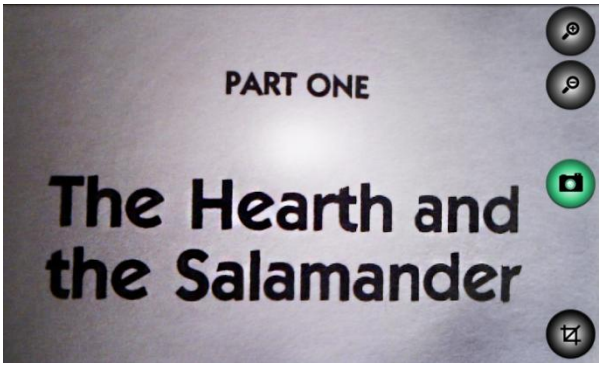


Figura 58 - UI - Captura de la imatge

La figura anterior mostra la imatge capturada, just un moment abans de ser enviada per al processat corresponent. Com es pot veure, la imatge ha estat enfocada correctament, i el flash a estat activat, ja que les condicions de llum no eren les idònies. Un cop capturada la imatge, l'aplicació mostra una petita notificació informant al usuari que s'està processant la imatge. La següent figura mostra aquesta notificació, aquesta és necessària, ja que el procés pot trigar varis segons, depenent de les regions trobades en la imatge.

Figura 59 - UI - Processant la imatge

Com hem vist en l'*apartat 5.4.3,* el processat és cridat per una classe asíncrona, això es utilitzat en aquest cas perquè, mentre s'està processant la imatge en segon pla, en la interfície d'usuari, es mostra la notificació de la figura anterior dinàmicament, ja que mostrant una mica de moviment en la UI, l'espera de l'usuari es més amena, i se li assegura que no a ocorregut cap error o que l'aplicació no s'ha congelat.

Un cop processada la imatge i rebuda la informació de tornada, l'aplicació mostra la informació a l'usuari com es mostra en la següent figura.

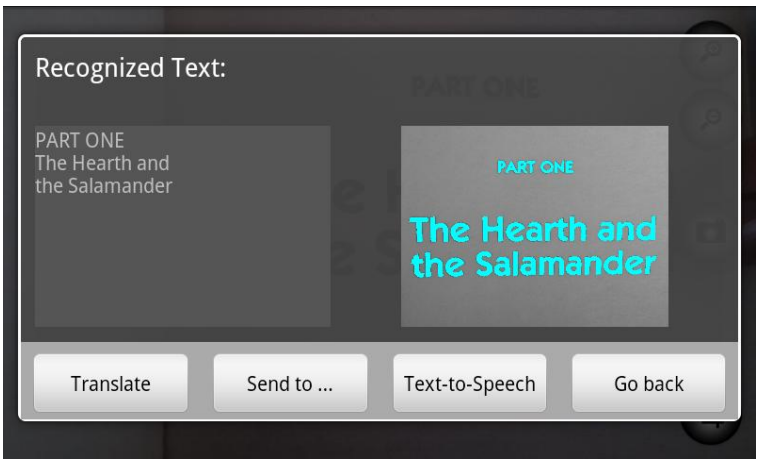


Figura 60 - UI - Resultats I

Com es pot apreciar, a part del propi text, motiu de l'aplicació, també es mostra la imatge processada amb el text reconegut marcat, per a que així, l'usuari pugui entendre millor els resultats obtinguts.

Per últim el botó de la part inferior de la *figura 56*, es el botó de "*crop*". El "*cropping*" es retallar una imatge per a centrar-nos en una part especifica. En aquest projecte això pot ajudar a enquadrar el text que es vol reconèixer, i així obtenir millors resultats. La següent figura mostra el procés del "*cropping*" un cop premut el botó corresponent.

Figura 61 - UI - "Cropping" d'una imatge

Com es pot veure en la figura anterior, el botó de "*crop*", mostra un rectangle en la pantalla. Aquest rectangle es pot canviar de mida estirant qualsevol dels seus vèrtex (en blanc) o costats, per a encabir la part desitjada de la imatge.

Un cop es té la part desitjada correctament enquadrada, el procés es exactament igual al explicat anteriorment, un cop premut el botó d' obtenir la imatge, el procés començarà, això si, en comptes de la imatge completa, només treballarem amb el requadre triat.

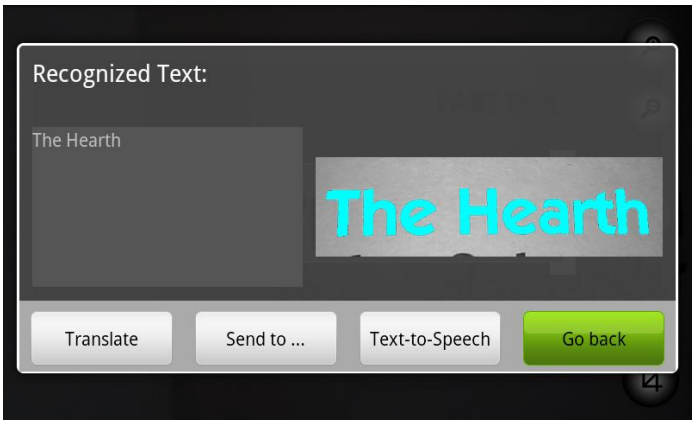


Figura 62 - UI - Resultats II

La figura anterior mostra els resultats obtinguts del reconeixement de la imatge correctament retallada. La sèrie de botons en la part baixa de la figura, s'usen per a les diferents accions que es poden fer amb el text, i es troben explicades en el següent apartat.

Un cop utilitzat el text i en disposició d'obtenir una nova imatge, clicarem el botó "*Go back*" el qual ens retornarà a la situació inicial de l'aplicació. Alternativament es pot usar el botó físic del dispositiu per anar enrere.

5.4.6. Ús del text reconegut

Un cop obtingut el text reconegut, es pot usar aquest per a infinitat de coses, només s'ha de copiar i fer servir com més ens convingui. No obstant, l'aplicació facilita les següents accions a dur a terme amb el text que acabem de reconèixer:

En primer lloc, una acció molt útil és, la traducció. Si, per exemple, el que s'acaba de reconèixer es un text d'un llibre en anglès, però el que es vol és el seu nom en català, l'aplicació el traduirà tot prement el botó "*Translate*", que es pot veure situat a la esquerra de la *figura 62*. En la següent figura es mostra aquest cas particular que s'acaba d'explicar.

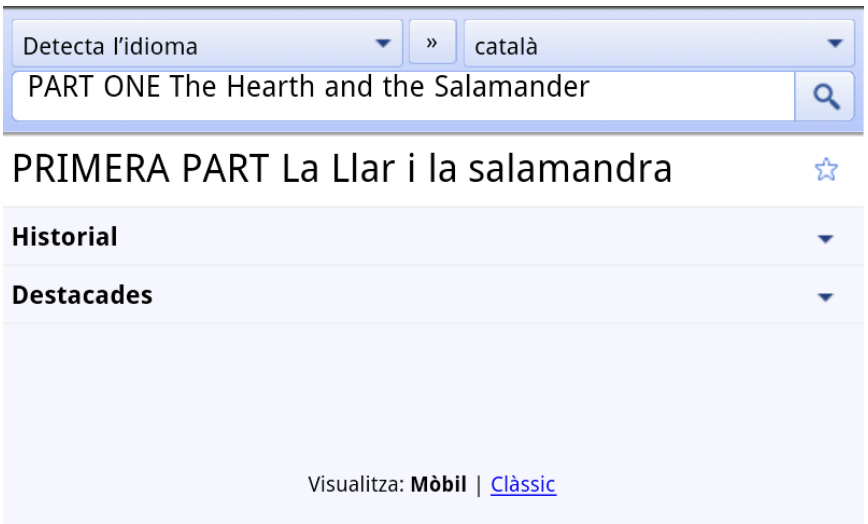


Figura 63 - UI - Traducció

Com que s'utilitza *Google Translate*, es possible traduir en ambdós sentits a tots els idiomes que aquest té disponibles, en aquest moment, 64 idiomes.

Un altre ús que agilitza la aplicació és, l'enviament del text a xarxes socials, direccions de correu electrònic, sistemes d'emmagatzematge en el núvol, etcètera. És a dir, permet copiar a qualsevol d'aquest serveis el text desitjat, per a compartir-ho amb qui es necessiti en qualsevol moment. La següent figura mostra algunes de les opcions disponibles, les quals dependran del software i hardware de cada usuari i dispositiu.

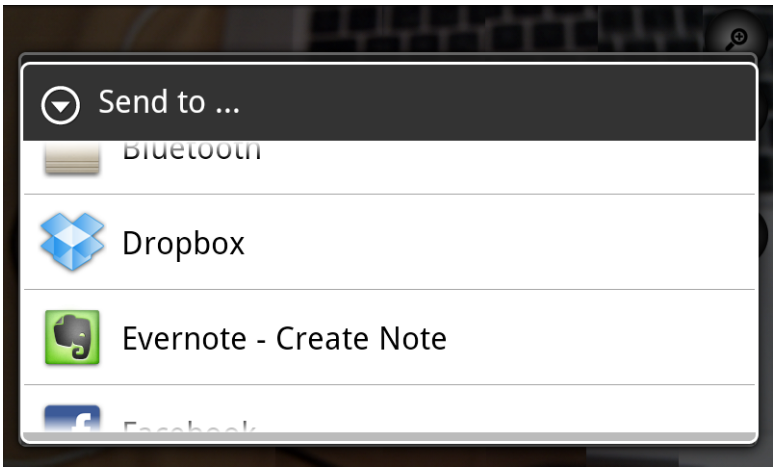


Figura 64 - UI - Enviar a ...

Per exemple, es pot enviar el text reconegut a un company, mitjançant la seva adreça electrònica, com es mostra en la següent figura, on l'aplicació ja ha generat l'assumpte i el cos del missatge, i per tan a l'usuari només li resta seleccionar l'adreça on enviar-lo.

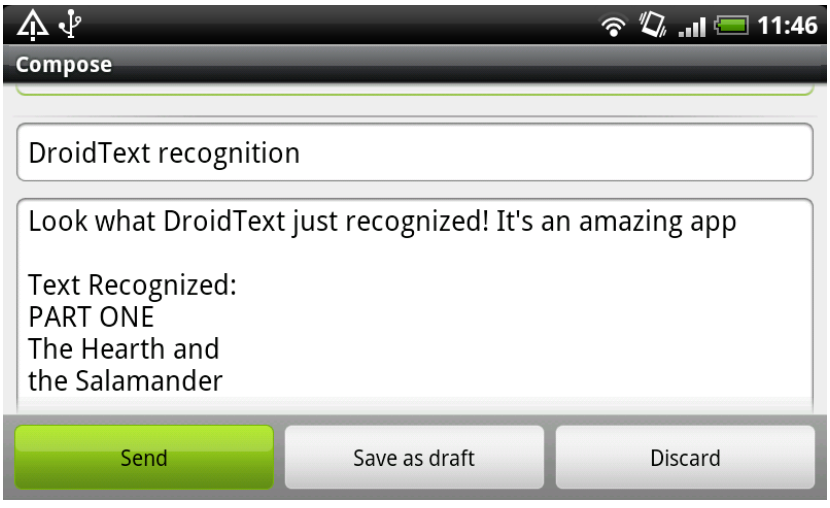


Figura 65 - UI - Enviar e-mail

Per acabar, l'ultima de les accions es el text a veu. Per a persones amb dificultats visuals de qualsevol tipus, es pot, en comptes de mostrar el text per pantalla, fer que el text sigui "parlat" pel dispositiu.

Per a això s'haurà d'escollir un dels idiomes suportat per la parla/reconeixement de veu d'*Android*, els quals són: anglès en les seves variants britànica i americana, castellà, francès, alemany i italià. Per a poder seleccionar aquests, en qualsevol moment abans de reconèixer el text, prement la tecla física "*Menú*" del dispositiu mòbil, apareixeran una sèrie de botons, en els quals podrem seleccionar l'idioma desitjat, com es mostra en la següent figura.

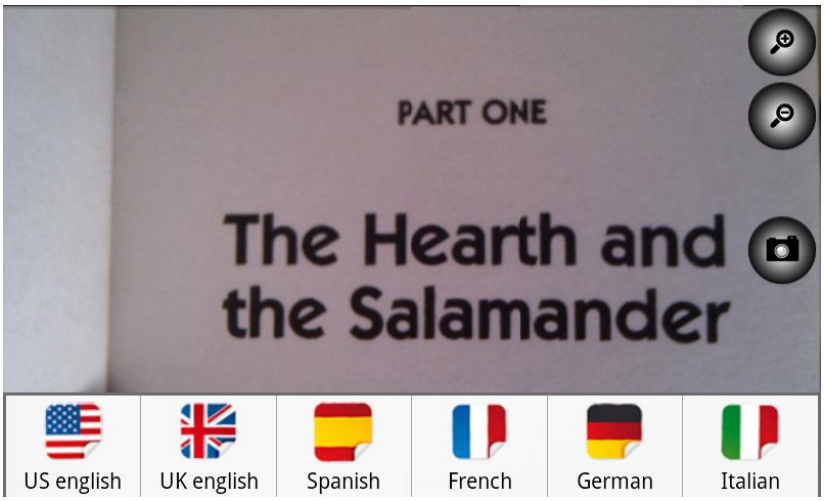


Figura 66 - UI - Sel·lecció d'idioma

Un cop seleccionat l'idioma desitjat, o per defecte l'anglès britànic, si es prem el botó "*Text-to-Speech*" l'aplicació "parlarà" el text reconegut.

6. Resultats

Per a realitzar els experiments s'ha usat la base d'imatges de la *ICDAR* del 2003 [31]. Aquesta base de dades, és especifica per a lectura i localització de text robust. A continuació s'explica com s'ha usat la base de dades completa en un principi, i un subgrup d'imatges més tard, les quals compleixen certes restriccions d'il·luminació, enfoc i contrast del text.

El set de proves conté 235 imatges de diferents resolucions, mides, il·luminació, enfoc i contrast obtingudes en diversos llocs, tan interiors com exteriors. En aquestes imatges s'ha observat que existeixen 3284 caràcters.

Figura 67 - Exemples d'imatges del set de proves

Per tal de provar el nostre algoritme amb més certesa s'ha creat un subset d'imatges en que les condicions d'il·luminació, contrast, textura, etc. són mínimament acceptables, seguint les restriccions que ja s'havien imposat.

El subset està composat per 71 imatges de la base de imatges inicial. S'han seleccionat aquelles que no contenen caràcters escapçats per reflexos i que estan mínimament enfocades i contrastades.

Per a realitzar les proves de detecció, es passarà a l'algoritme les imatges del set de prova triat. Per a cada una de les imatges, es guardarà una nova imatge (o grup d'imatges si la imatge conté diverses línies de text) amb el resultat del algoritme, per a després comprovar l'encert d'aquest. Un cop calculat el ràtio de detecció, s'usaran aquestes noves imatges com a entrada del OCR per a comprovar l'eficàcia d'aquest, amb aquestes imatges.

6.1. Detecció de text

A part d'imatges com les de la *figura 67*, és a dir, imatges del mon real, sense cap tipus de control, però ben enfocades i amb els caràcters llegibles, s'ha trobat que el set d'imatges conté també, imatges "defectuoses". En els següents punts se'n mostren alguns exemples:

- Imatges amb reflexos:

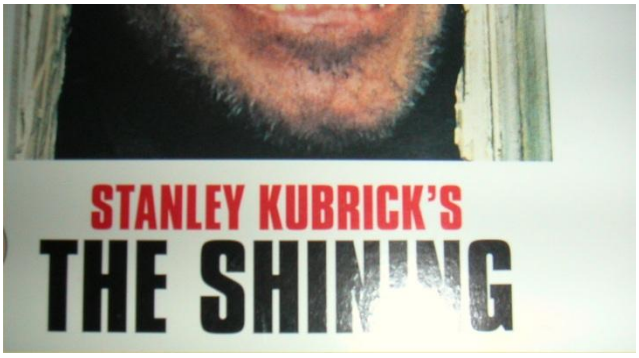


Figura 68 - Imatge amb reflex

Com es pot veure en la imatge els caràcters *'NIN'* tenen un reflex i per tant el nostre algoritme no serà capaç de reconstruir aquest caràcter. A més inclús la lletra *'G'* la qual sembla que es veu correctament a ull nu, es confon amb el fons.

- Imatges amb poc contrast o doble contrast

Figura 69 - Imatge amb doble contrast

En la imatge es pot apreciar que els caràcters més petits gairebé no tenen contrast i per tant no es podran detectar. El text més gran conté un doble contrast el qual el nostre algoritme no podrà detectar com a lletra.

- Imatges amb massa textura

Figura 70 - Imatge texturada en excés

Aquest tipus d'imatges no deixen obtenir els contorns amb claredat ja que tenen contorns dins les mateixes lletres i per tan l'algoritme no serà capaç de detectar-les.

- Imatges desenfocades

Figura 71 - Imatge desenfocada

Caràcters com la *'H'* en la imatge són difícils de detectar ja que al no estar correctament enfocada el contrast no es el correcte per a detectar amb claredat.

En la següent taula es pot comprovar l'eficàcia del algoritme durant la fase de desenvolupament:

	Nombre d'imatges	*Caràcters presents*	*Caràcters reconeguts*	*Falsos positius*
Imatges originals	235	3284	52,4 %	375
Selecció d'imatges	71	1282	93,9 %	88

Taula 5 - Resultats del test de l'algoritme

Dels 3284 caràcters que el set conté, només un 52,4% són reconeguts com a text, pel nostre algoritme. A part del text reconegut l'algoritme ha deixat 375 falsos positius, els quals el motor OCR podria confondre amb text (si es prou similar a algun caràcter).

Aquest baix percentatge d'encert es degut al tipus d'imatges dins del set, com ara les imatges tipus que acabem de mostrar. Per això tornarem a executar l'experiment, però ara fent servir el subset de proves que s'ha creat.

Aquest subgrup d'imatges conté 1282 caràcters dels quals, ara es reconeixen un 93,9% en comparació del 50% i escaig, que s'obtenia abans. A més, s'ha reduït el nombre de falsos positius a només 88, això es menys de dos falsos positius per imatge.

Els pocs caràcters que no es detecten, són aquells que tot i que la imatge està ben contrastada, conté algun caràcter poc contrastat o mal enfocat i per tant no el detectem correctament, o degut a algun fals negatiu del nostre classificador. Alguns exemples d'aquests errors són mostrats a continuació:

- Caràcters continus

Figura 72 - Caràcters continus

En alguns casos caràcters de tipografia separada, és poden ajuntar com és el cas de la *'ST'* de la figura anterior.

- Falsos negatius

Figura 73 - Fals negatiu

En comptades ocasions el nostre classificador dona falsos negatius, regions que hauria de reconèixer com a caràcter les descarta, això pot ser degut a formes estranyes d'algun caràcter, com la *'J'* majúscula de la *figura 73*, que ha estat descartada perquè es massa allargada per a ser una lletra, en comparació amb les altres.

- Zones mal enfocades

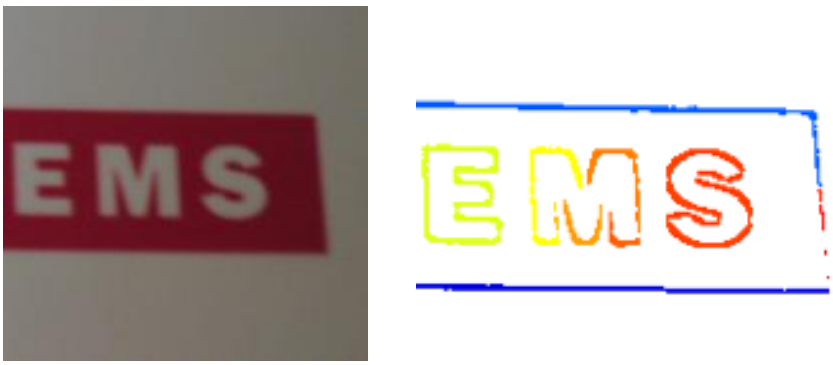


Figura 74 - Caràcter erroni

En alguns casos, la imatge en general sembla ben enfocada, però alguna zona d'aquesta ha quedat borrosa. Com es pot apreciar en la *figura 74* la lletra *'M'* no ha quedat correctament segmentada. S'han obtingut diversos components connexes, i la recuperació de la forma original, no ha pogut posar-hi remei, ja que la zona original no es de gran ajuda.

6.2. Reconeixement de caràcters (OCR)

Per al reconeixement dels caràcters s'ha usat el OCR comercial *Tesseract*, el qual al passar-li la imatge correctament segmentada obté els caràcters corresponents, codificats en *ASCII*. La següent taula mostra els resultats obtinguts:

	Caràcters presents en la imatge	Falsos Positius presents en la imatge	Caràcters reconeguts
Imatges originals	1721	375	81 %
Selecció d'imatges	1203	88	94,6 %

Nota: Per a contar l'eficàcia de l'OCR només s'han tingut en compte els caràcters detectats anteriorment

Taula 6 - Resultats del test del OCR

Del set d'imatges originals el OCR utilitzat, reconeix un 81% dels caràcters existents. En canvi, amb el subset d'imatges escollit, els caràcters reconeguts augmenten fins a un 94,6%. Això es degut en gran part, a l'eliminació de falsos positius i de lletres escapçades.

7. Gestió del projecte

7.1. Planificació

El projecte ha estat dividit en tot moment en dues parts, el desenvolupament dels algoritmes, i l'aplicació *Android*, en la següent taula es pot veure el temps i períodes en els que aquestes dues tasques s'han dut a terme.

	Mod de	Nombre de tarea	Duración	Comienzo	Fin	Pred	% completi
1		**Fase de desenvolupament**	**81 días**	**lun 12/09/11**	**lun 02/01/12**		**100%**
15		**Implementació de l'aplicació**	**52 días**	**mar 03/01/12**	**mié 14/03/12**		**100%**
23		Redacció de la memoria	150 días	mié 14/09/11	mar 10/04/12	2	100%
24		Presentació i demos	7 días	mié 11/04/12	jue 19/04/12	23	100%

Taula 7 - Terminis del projecte

La fase de desenvolupament conté més feina que la implementació de l'aplicació, ja que aquesta part conté la recerca d'algoritmes i les proves d'aquests. En la *taula 8* i la *taula 9* podem veure el desglossament de les dues fases.

	Mod de	Nombre de tarea	Duración	Comienzo	Fin	Pred	% completi
1		**Fase de desenvolupament**	**81 días**	**lun 12/09/11**	**lun 02/01/12**		**100%**
2		Recerca de Datasets	2 días	lun 12/09/11	mar 13/09/11		100%
3		**Detecció de text**	**35 días**	**mar 13/09/11**	**lun 31/10/11**		**100%**
4		Recerca sobre l'estat de l'art	14 días	mar 13/09/11	vie 30/09/11	2	100%
5		Implementació dels metodes trobats	14 días	lun 03/10/11	jue 20/10/11	4	100%
6		Proves dels algortimes	7 días	vie 21/10/11	lun 31/10/11	5	100%
7		**Filtrat de regions**	**21 días**	**mar 01/11/11**	**mar 29/11/11**		**100%**
8		Recerca sobre l'estat de l'art	7 días	mar 01/11/11	mié 09/11/11	6	100%
9		Implementació del filtratge	14 días	jue 10/11/11	mar 29/11/11	8	100%
10		**Millores a l'algoritme**	**9 días**	**mié 30/11/11**	**lun 12/12/11**		**100%**
11		Recerca de metodes	1 día	mié 30/11/11	mié 30/11/11	9	100%
12		Implementació de la millora de regions	6 días	jue 01/12/11	jue 08/12/11	11	100%
13		Separació de linies de text	2 días	vie 09/12/11	lun 12/12/11	12	100%
14		Test del algoritme	45 días	mar 01/11/11	lun 02/01/12	6	100%
15		**Implementació de l'aplicació**	**52 días**	**mar 03/01/12**	**mié 14/03/12**		**100%**
23		Redacció de la memoria	150 días	mié 14/09/11	mar 10/04/12	2	100%
24		Presentació i demos	7 días	mié 11/04/12	jue 19/04/12	23	100%

Taula 8 - Tasques fase de desenvolupament

Com es pot veure en aquestes figures, la redacció de la memòria s'ha fet de forma continuada, per poder anar explicant les solucions emprades i els errors trobats en el moment que passen, i no perdre'n detalls.

Per a cada secció de la fase de desenvolupament, s'ha tingut una etapa de recerca sobre l'estat de l'art, per a estudiar i provar diferents solucions al problema.

La següent taula mostra les tasques dutes a terme en el desenvolupament de l'aplicació *Android*.

	Mod de	Nombre de tarea	Duración	Comienzo	Fin	Pred	% completi
1	✓	**Fase de desenvolupament**	**81 días**	**lun 12/09/11**	**lun 02/01/12**		**100%**
15	✓	**Implementació de l'aplicació**	**52 días**	**mar 03/01/12**	**mié 14/03/12**		**100%**
16	✓	Recerca sobre Android	2 días	mar 03/01/12	mié 04/01/12	14	100%
17	✓	Desenvolupament de l'aplicació basica	7 días	jue 05/01/12	vie 13/01/12	16	100%
18	✓	Reproducció del algoritme	14 días	jue 12/01/12	mar 31/01/12	17	100%
19	✓	Inserció del OCR	2 días	mié 01/02/12	jue 02/02/12	18	100%
20	✓	Optimització de l'aplicació	14 días	vie 03/02/12	mié 22/02/12	19	100%
21	✓	Test de la reprdoucció del algoritme	21 días	mié 01/02/12	mié 29/02/12	18	100%
22	✓	Test de l'Aplicació	50 días	jue 05/01/12	mié 14/03/12	16	100%
23	✓	Redacció de la memoria	150 días	mié 14/09/11	mar 10/04/12	2	100%
24	✓	Presentació i demos	7 días	mié 11/04/12	jue 19/04/12	23	100%

Taula 9 - Tasques desenvolupament de l'aplicació

El desenvolupament real del projecte ha seguit les pautes dictades en aquests terminis amb gran fidelitat, normalment amb uns desviaments de més/menys 2 dies, sense afectar el treball del conjunt de tasques.

Una de les tasques que més s'ha desviat del programa és la millora de regions, tant en la fase de desenvolupament com en la seva reproducció en l'aplicació. Això és degut a la complexitat de l'algoritme, del qual no s'havia reconegut la seva dificultat en un principi.

Tot i això com ja s'ha dit, aquesta desviació no ha comportat el desplaçament de tasques posteriors, ja que algunes tasques han durat algun dia menys del esperat.

En la *figura 75* i la *figura 76* podem veure els diagrames de Gannt que detallen les tasques dutes a terme.

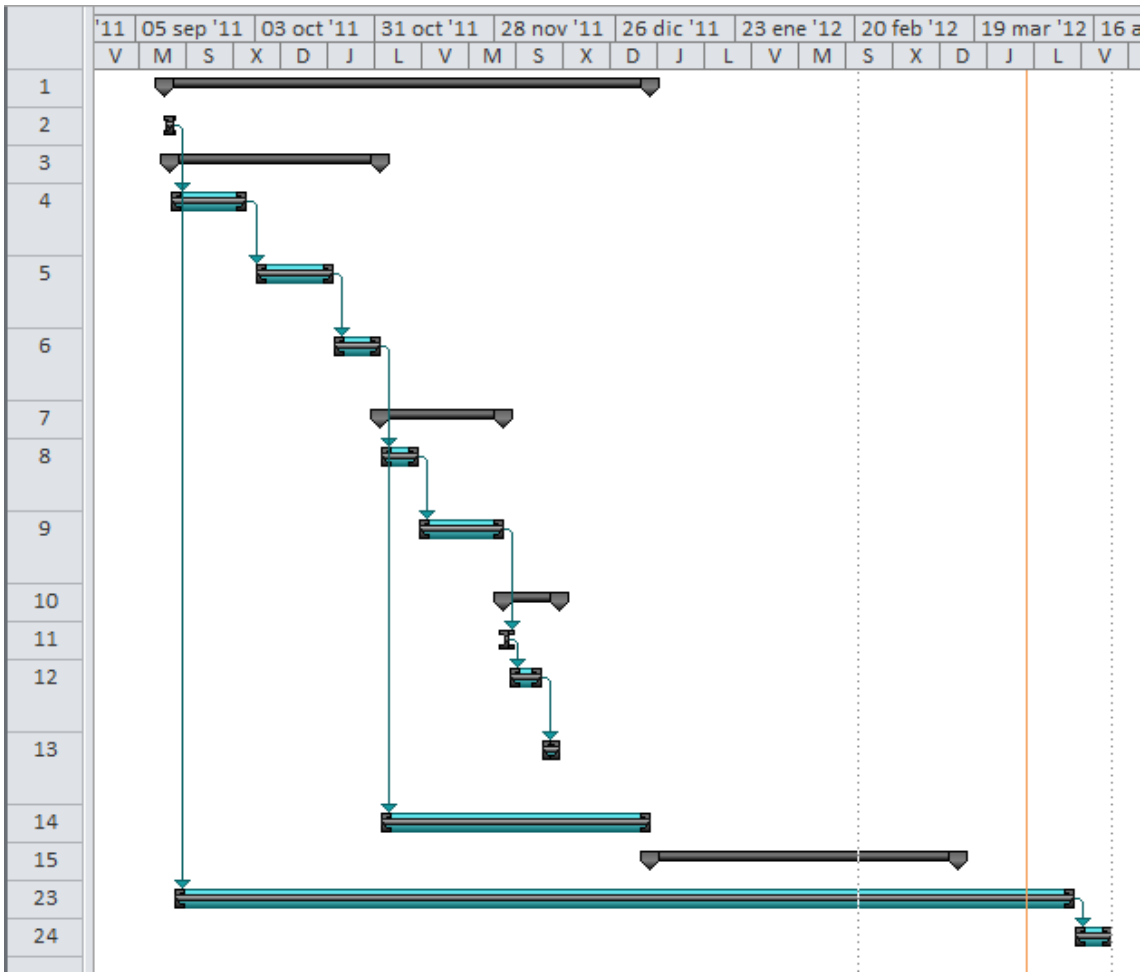


Figura 75 - Diagrama de Gannt I | Desenvolupament

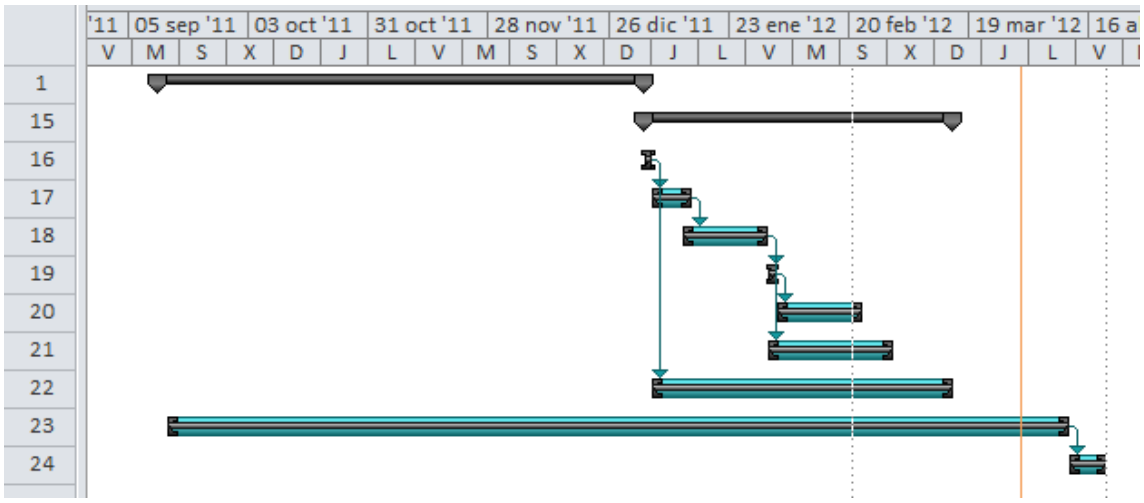


Figura 76 - Diagrama de Gannt II | Aplicació

7.2. Pressupost

En aquest apartat s'exposa el cost del projecte. S'ha assumit una jornada laboral amb mitja de 7 hores per dia. El cost del projecte es divideix en tres parts:

- ***Recursos Humans:***

En la següent taula, es pot veure el cost total del projecte en recursos humans. Els dies treballant són, el nombre total de dies que s'ha dedicat en aquella àrea, però s'ha de ponderar per la carrega de treball de cada tasca per a obtenir les hores reals, en les quals s'ha treballat. Per exemple la documentació es fa cada dia però només un 10% d'aquest, en canvi, en recerca usem un 90% de les hores de treball del dia.

Concepte	*Dies treballant*	*Càrrega de temps*	*Hores*	*Preu per hora*	*Cost*
Recerca	30	90 %	150	35 €	5.250 €
Desenvolupament	80	80 %	370	52 €	19.240 €
Proves	120	40 %	350	50 €	17.500 €
Documentació	160	10 %	90	20 €	1.800 €
Total			960		43.790 €

Taula 10 - Cost en recursos humans

- ***Recursos Hardware:***

El hardware utilitzat es un portàtil MacBook Pro amb un cost aproximat de 2100€, el qual no s'utilitza només per al projecte, per tant n'haurem d'extreure l'amortització d'aquest per al projecte. Si tenim en compte que la vida útil d'un ordinador es 5 anys i aquest es fa servir 8 hores al dia durant 22 dies al mes, això ens deixa amb el següent cost per hora:

$$Cost_{Hardware} = \frac{2100\ €}{5\ anys * 12\ mesos/any * 22\ dies/mes\ * 8\ hores/dia} = 0{,}199\ €/hora$$

Hem estat en tot moment usant l'ordinador durant cada una de les fases, el que ens deixa segons la taula 10 amb 960 hores d'ús, per tant:

$$Cost_{Hardware} = 0{,}199\ €/hora\ * 690\ hores\ =\ 137{,}31\ €$$

Igualment podem calcular el cost del dispositiu mòbil un HTC Desire amb un preu aproximat de 500 € i un ús aproximat de 400 hores per a aquest projecte.

$$Cost_{Mòbil} = 0{,}047\ €/hora * 400\ hores = 18{,}94\ €$$

- ***Recursos Software:***

Igual que en el cas del hardware, el software no s'utilitzarà només per aquest projecte per tant haurem de calcular el preu amortitzat pel projecte, tenint en compte que la vida útil del software es considera de dos anys. La següent taula mostra el cost dels diferent softwares utilitzats, exceptuant els open source.

Software	*Preu*
MATLAB 7.12	1600 €
Image Processing Toolbox	750 €
MS Project 2010	775 €

Taula 11 - Cost del software

Si apliquem una formula semblant a la utilitzada per a calcular el hardware, tenint en compte que Matlab s'ha usat unes 600 hores i MS Project unes 8 hores obtenim:

$$Cost_{Software} = 0{,}379\ €/hora * 600\ hores = 227{,}273\ €$$

$$Cost_{MS\ Project} = 0{,}183\ €/hora * 5\ hores = 1{,}464\ €$$

Un cop calculat el cost de les tres parts del projecte obtenim que el cost total d'aquest és de 44.175 €.

Software	*Preu*
Cost en Recursos Humans	43.790 €
Cost del Hardware	156,25 €
Cost del Software	228,74 €
Total	44.175 €

Taula 12 - Cost total del projecte

8. Conclusions

8.1. Conclusió del projecte

Els resultats obtinguts, són molt similars a aquells esperats. S'ha obtingut una aplicació per a dispositius mòbils *Android* capaç de reconèixer text en escenes naturals, que funciona amb suficient eficàcia.

L'aplicació és capaç d'obtenir una imatge amb la càmera del dispositiu, binaritzar-la, i sobre aquesta, detectar i eliminar les zones que no pertanyen a cap caràcter. Un cop fet això, l'aplicació és capaç de proporcionar la nova imatge a un OCR el qual retorna el text reconegut.

Amb aquest text reconegut, es poden fer tres accions bàsiques, traduir-lo, enviar-lo a traves de la xarxa, o escoltar-lo.

Sobre l'eficàcia del algoritme s'ha vist que es pot obtenir de promig un 94% dels caràcters presents en la imatge, si es respecten les restriccions acordades. Un cop l'algoritme traslladat a l'aplicació *Android* el seu comportament no ha disminuït en excés. No és senzill calcular quins percentatges s'obtenen amb el dispositiu mòbil, ja que no podem passar a l'aplicació un set de proves perquè l'executi, però amb les proves realitzades, el comportament ha esdevingut molt similar.

Cal fer notar que la captura de la imatge es assistida per l'usuari, és a dir, es ell qui capturarà la imatge, i per tant, pot tenir cura de les restriccions plantejades. En el pitjor dels casos aquest pot repetir la captura si el resultat no és l'esperat.

Per tant globalment, es pot dir que els objectius han estat complerts, amb resultats satisfactoris, i l'aplicació respon amb el que se li demana.

8.2. Treball futur

En un futur, una branca en la que segur s'haurà de treballar, és en l'entrenament de l'arbre de decisió, amb imatges les quals continguin almenys totes les tipologies estàndard de text en diferents mides, per així millorar l'eficiència de l'algoritme de reconeixement.

Com a altres millores a aportar al projecte, amb més temps en un futur, s'hauria de suprimir, en la mesura del possible, algunes de les restriccions que aquest projecte s'ha imposat. Com ja s'ha comentat, l'aplicació conté aquestes restriccions bàsiques: les imatges no poden ser escapçades, text sense rotació, els caràcters no poden ser molt petits, ni de escriptura continua.

Per a la primera, s'hauria de crear un algoritme de reconstrucció de caràcters, el problema rau, en el consum de recursos d'aquests algoritmes, i per tant són impracticables per a dispositius mòbils, en aquest moment.

Acceptar rotació del text és un altra possible millora, no en quant al OCR, ja que aquest només accepta text en format horitzontal, però si crear un algoritme capaç de dividir la imatge en paraules i comprovar-ne l'orientació per a rectificar la imatge i passar-la al OCR.

Per últim, respecte a l'escriptura continua, no podem fer-hi res si estem utilitzant un OCR comercial, per tant la solució seria crear un OCR dedicat a escriptura continua, cosa que donaria lloc a un altre projecte.

En canvi, per a caràcters petits tenim una solució més sencilla. Per a poder detectar-los s'haurà de crear un altre arbre de decisió, el qual s'haurà d'entrenar amb aquest tipus de lletres més petites, a més de canviar alguns dels paràmetres del algoritme, com la mida mínima de una regió. Això pot donar possibles problemes amb la tipologia que ara detectem amb facilitat, però ja ens en preocuparem quant continuem el treball.

8.3. Possibilitats de mercat

Com s'ha explicat al llarg del document, les aplicacions mòbils per a reconeixement de text estan en una fase molt prematura, ja que aquelles que funcionen en el propi dispositiu mòbil, i no en un servidor, tenen un percentatge d'encert no gaire elevat.

En el cas de l'aplicació que ens ocupa, l'eficàcia, millora a les presents aplicacions que processen la imatge en el propi dispositiu, en el cas que es compleixin les restriccions esmentades. En canvi, algunes d'elles com *Google Goggles* que processa la imatge en els servidors, supera amb escreix tan l'àrea d'abast com l'eficàcia d'aquesta aplicació.

Un dels grans problemes de les aplicacions de reconeixement de text, és l'estat del art dels OCR, que encara que han millorat molt últimament, no n'hi ha cap que pugui reconèixer text amb gairebé un 100% de encert. Altres problemes d'aquest tipus d'aplicació, en concret per a dispositius mòbils, és la qualitat de la imatge.

Al fer ús d'una càmera de 5 megapixels o similar, sense cos de càmera, degut a que els mòbils són dissenyats per a ser lleugers, i les vibracions ocasionades per l'usuari, poden comportar trepidacions i parts borroses que perjudiquen als algoritmes de binarització. A més, és clar, que al no tenir un entorn controlat, les condicions d'il·luminació poden ser totalment oposades a les desitjades.

Tot i això, gracies al algoritme dissenyat en aquesta memòria, que intenta millorar la imatge, les prestacions de l'aplicació augmenten el suficient per a que, el producte actual pugui ser comercialitzat en una versió de prova, fins a poder aconseguir un dels objectius dels treballs futurs abans proposats, i poder tenir una aplicació perfectament funcional, que faciliti la vida dels usuaris.

8.4. Valoració personal

Per a mi aquest projecte, ha significat poder endinsar-me més en dos àrees que són molt interessants per a mi: la visió per computador i les aplicacions destinades a dispositius mòbils.

La robòtica sempre ha sigut una de les meves passions, i una part d'ella podríem dir que és la visió per computador. En aquesta àrea he aprés, gracies al projecte, camps que no s'expliquen en assignatures de la carrera com poden ser, per exemple, els motors OCR, i he aprofundit en d'altres com pot ser el processament d'imatges, que tot i haver-se explicat en alguna assignatura com ara Visió per Computador, el nivell de profunditat assolit amb el projecte és més elevat.

Respecte a dispositius mòbils, sempre havia tingut la curiositat de programar una aplicació per al meu dispositiu Android, però mai havia tingut temps de fer-ho. El projecte ha estat la meva oportunitat d'or per començar a desenvolupar per dispositius mòbils, i l'experiència ha estat tant bona, que de ben segur que ho seguiré fent en un futur.

Respecte al resultat del projecte, em sento molt satisfet de la feina feta i del seu resultat. Tant és així que, estic fent els tràmits pertinents per a publicar l'aplicació a *Google play*, la tenda d'aplicacions de *Google* per a *Android*.

9. Referències

[1] Gartner, Inc. (2007-2011). *Gartner smartphone marketshare.*

[2] OmniPage - *http://www.nuance.com/omnipage*

[3] ABBYY FineReader - *http://finereader.abbyy.com/*

[4] ReadIris - *http://www.irislink.com/readiris*

[5] Prizmo - *http://www.creaceed.com/prizmo/iphone/*

[6] Word Lens - *http://questvisual.com/*

[7] Google Goggles - *http://www.google.com/mobile/goggles/*

[8] J. Fabrizio, B. Marcotegui, M. Cord (2009). *Text segmentation in natural scenes using toggle-mapping.*

[9] B. Epshtein, E. Ofek, Y. Wexler (2010). *Detecting text in natural scenes with stroke width transform.*

[10] J. Canny (1986). *A computacional approach to edge detection.*

[11] T. Retornaz, B. Marcotegui (2007). *Scene text localization based on the ultimate opening.*

[12] P. Clark, M. Mirmehdi (2000). *Finding text regions using localised measures.*

[13] S. Muhammad, L. Prevost (2007). *Texture based text detection natural scene images: a help to blind and visually impaired persons.*

[14] GOCR - *http://jocr.sourceforge.net/*

[15] Ocrad - *http://www.gnu.org/software/ocrad/*

[16] Test Ocrad - *http://www.mscs.dal.ca/~selinger/ocr-test/*

[17] Tesseract - *http://code.google.com/p/tesseract-ocr/*

[18] R. Smith –Google Inc. (2007). *An overview of the Tesseract OCR engine.*

[19] MATLAB 7.13 (R2011a) - *http://www.mathworks.es/products/matlab/*

[20] TesseractOCR - *http://www.malcolmhardie.com/ocr/xcode4.html*

[21] Weka 3 - *http://www.cs.waikato.ac.nz/ml/weka/*

[22] Eclipse Indigo - *http://www.eclipse.org/*

[23] Java SDK - *http://www.oracle.com/technetwork/java/javase/overview/index.html*

[24] Android SDK r16 - *http://developer.android.com/sdk/index.html*

[25] Android 2.2 Platform - *http://developer.android.com/sdk/android-2.2.html*

[26] Android versions usage - *http://developer.android.com/resources/dashboard/platform-versions.html*

[27] ADT 16.0.1 - *http://developer.android.com/guide/developing/tools/adt.html*

[28] Android OpenCV - *http://opencv.willowgarage.com/wiki/Android*

[29] Android Tesseract - *http://code.google.com/p/tesseract-android-tools/*

[30] NDK r7 - *http://developer.android.com/sdk/ndk/index.html*

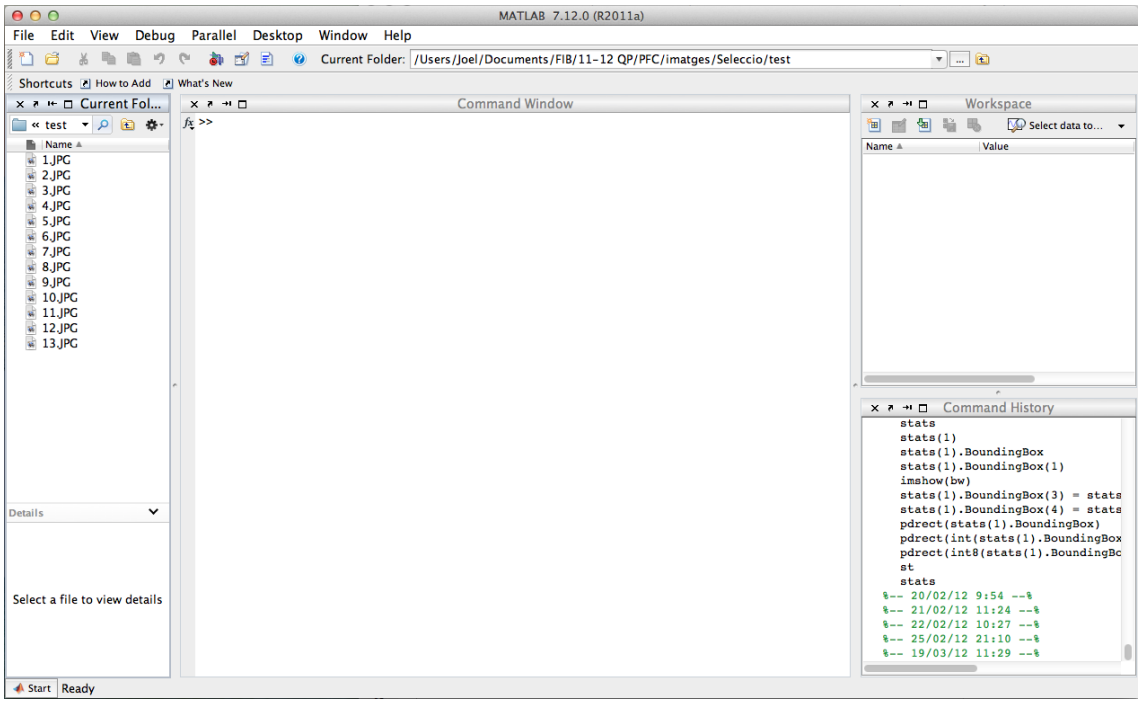
[31] ICDAR 2003 - *http://www.essex.ac.uk/csee/icdar2003*

10. Annexes

10.1. Instal·lació de l'entorn de treball

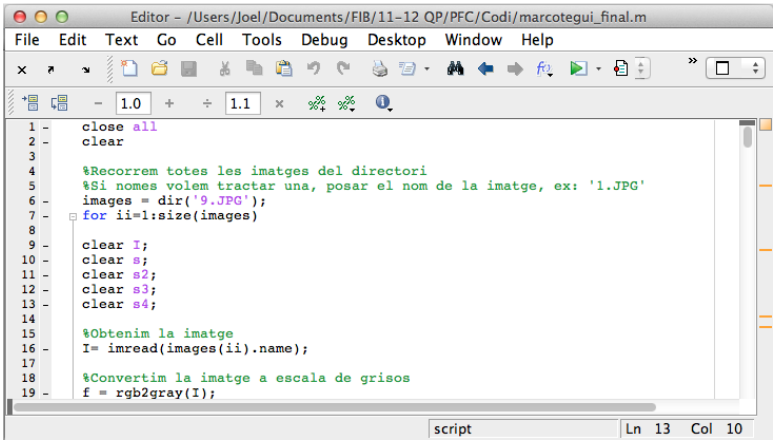
Desenvolupament del algoritme

Per al desenvolupament del algoritme s'ha usat Matlab, que és un entorn de treball integrat (IDE) creat per l'empresa MathWorks. Ni aquest, ni les seves toolbox són gratuïtes, però tenen una versió amb descompte per a estudiants, i la universitat disposa de una llicencia per a utilitzar-lo en l'àmbit acadèmic.



La pantalla principal de Matlab esta dividida en 4 parts a més dels menús. La primera es mostra a la esquerra i mostra la carpeta actual de treball. A la dreta hi ha l'espai de treball on mostra totes les variables amb les que estem treballant, i la historia de comandes que s'han usat. Per últim al centre hi ha la finestra de comandes, on es pot dir a Matlab que ens interpreti una instrucció.

Com que el mode de comandes directes és una mica carregós i no es pot guardar la seqüencia de instruccions executada, Matlab permet la creació de fitxers, amb tot d'instruccions que podran ser executades seqüencialment, amb un sol clic.



Desenvolupament de l'aplicació

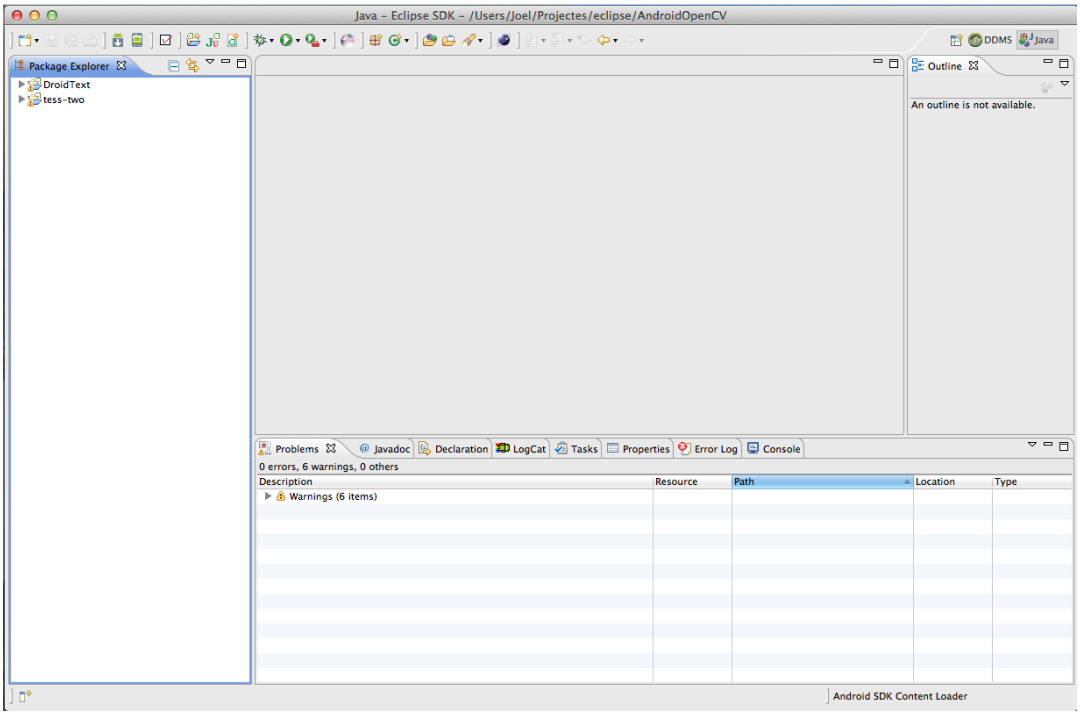
Per a desenvolupar l'aplicació es farà servir l'IDE Eclipse amb diversos pluguins i llibreries per a desenvolupar amb Java en Android.

Primer de tot caldrà descarregar l'IDE Eclipse de la següent pagina:

http://www.eclipse.org/downloads/packages/eclipse-classic-372/indigosr2

Per a poder executar Eclipse s'ha de tenir instal·lat el JDK que es pot trobar en la següent pagina:

http://www.oracle.com/technetwork/java/javase/downloads/jdk-7u3-download-1501626.html



L'IDE conté un "Package explorer" on observar l'estructura dels projectes, i diverses pestanyes per els costats totalment configurables, les quals poden mostrar des de la consola, a logs , JavaDocs o errors de compilació. Al centre es poden obrir els diversos fitxers a editar.

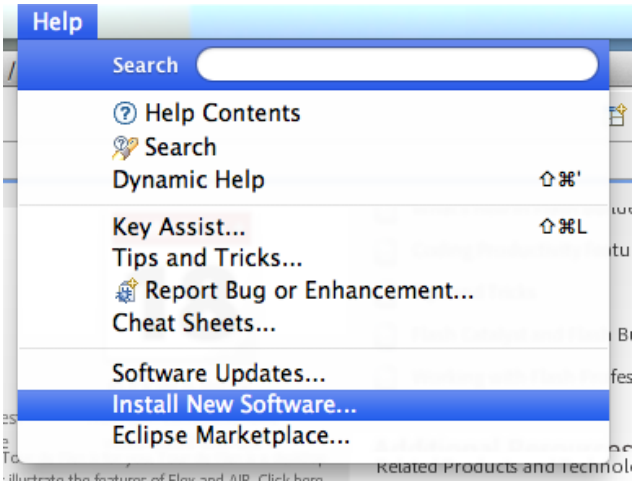
Un cop familiaritzats amb l'IDE s'haurà de descarregar l'Android SDK de la següent pagina:

http://developer.android.com/sdk/index.html

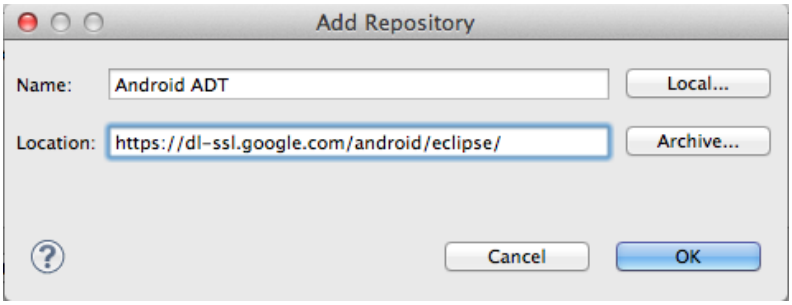
Les eines que s'acaben de descarregar s'han de col·locar en algun lloc del nostre sistema de fitxers conegut, ja que serà necessari en el següent pas.

El següent pas és instal·lar el plugin ADT per a Eclipse, el qual permetrà crear projectes, administrar diferents targets, etc.

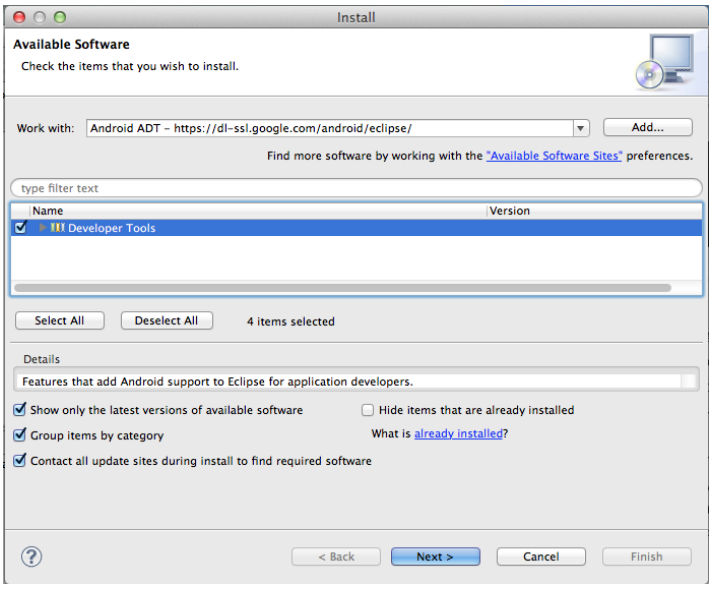
Primer s'haurà d' afegir el repositori de ADT en l'Eclipse i després descarregar-lo. Es clica "Help" --> "Install Software" --> "Add..".



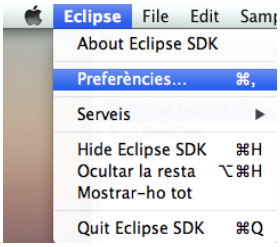
S'afegeix el següent repositori: https://dl-ssl.google.com/android/eclipse/



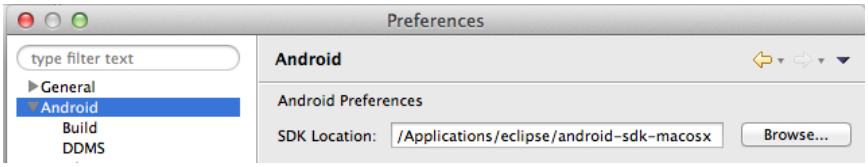
Es seleccionen les Developer Tools, i s'accepten tots els paràmetres per defecte. Un cop instal·lat el plugin haurem de reiniciar l'IDE.



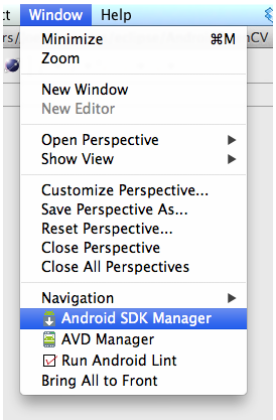
Ara s'ha de configurar el ADT perquè trobi el Android SDK descarregat, i descarregar els SDK per als "targets" específics. Es clica "Eclipse" --> "Preferencies".



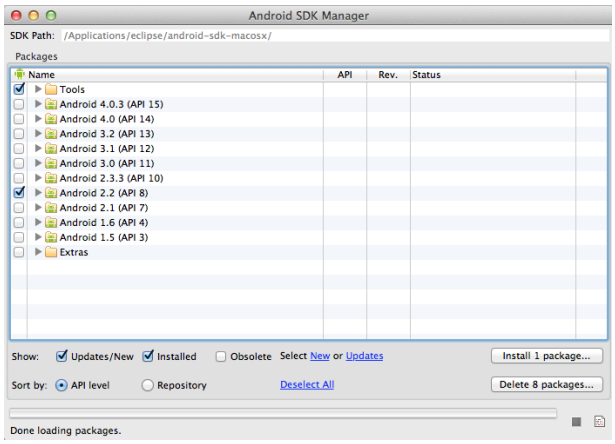
S'introdueix la ruta al directori on hem descarregat el Android SDK.



Ara ja es pot utilitzar l'ADT per a descarregar el SDK específic per al nostre "target". Es clica "Window" --> "Android SDK Manager".



Es selecciona el SDK específic que es necessita ,que en aquest cas, és el 2.2 (API 8) i s'instal·la.



Ara ja es pot crear aplicacions per a Android, debugar-les, crear UI i tot de forma senzilla i integrada en el mateix entorn.

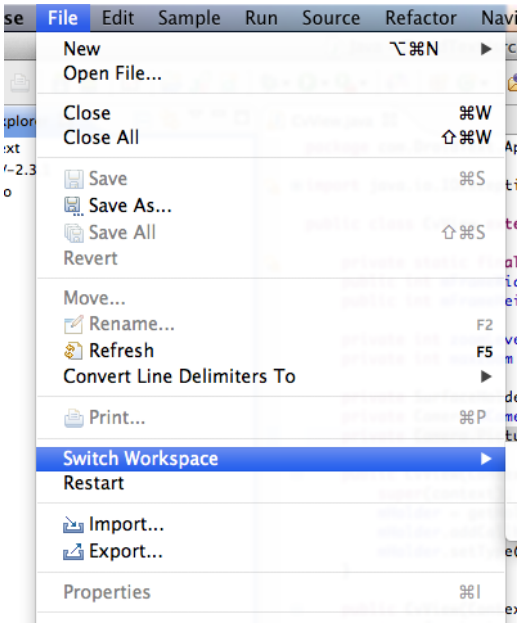
Per últim per al cas del projecte que ens ocupa, haurem de afegir les llibreries que volem utilitzar en el nostre espai de treball.

OpenCV

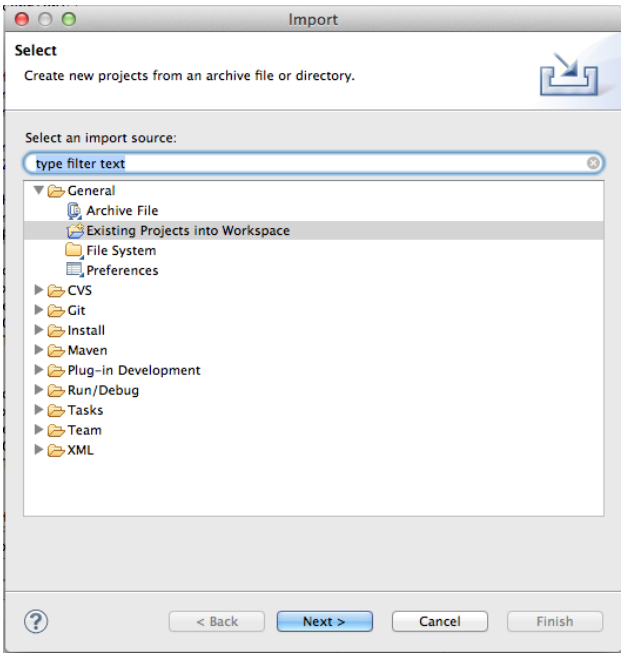
La llibreria de processat d'imatges, es pot descarregar d'aquest repositori:

http://sourceforge.net/projects/opencvlibrary/files/opencv-android/2.3.1/

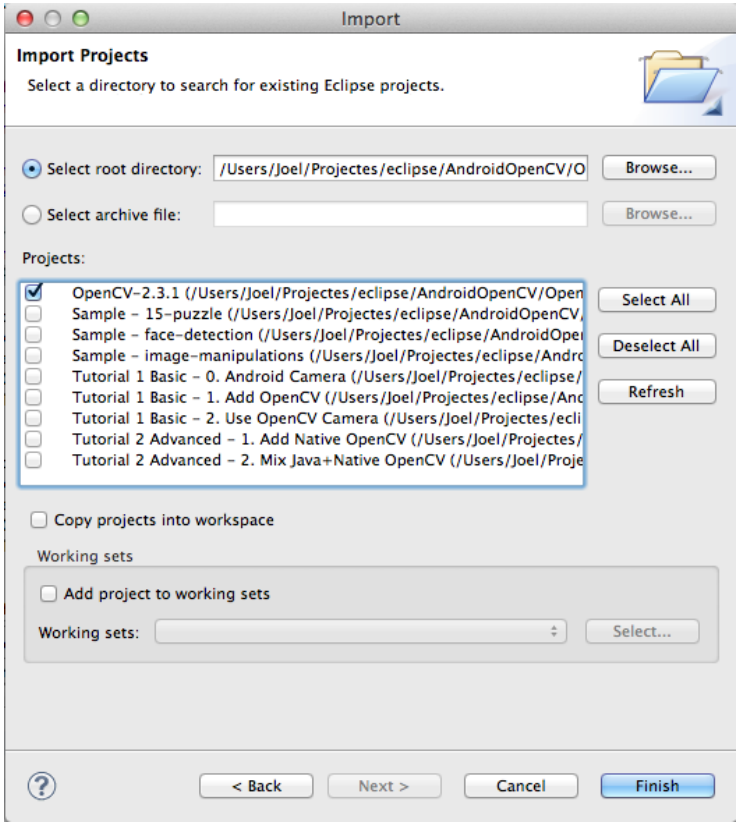
Un cop descarregada es pot desar en la carpeta del espai de treball que s'usarà a Eclipse. Si es necessari, es pot canviar l'espai de treball d'Eclipse clicant "File" --> "Switch workspace" --> "Other" i després seleccionar el directori que es vulgui.



Ara s'importarà la llibreria al nostre entorn de treball de l'IDE. Cliquem "File" --> "Import...". Seleccionem "Existing project into workspace"



A continuació indiquem el path al directori on hi ha la llibreria, i es selecciona OpenCV, la resta de projectes són d'exemple així que no cal seleccionar-los. Es clica "Finish" i ja hi haurà la llibreria correctament importada al workspace del IDE.



Tesseract OCR

Primer es descarregarà el Tesseract Tools for Android, però es descarregarà un fork en el qual s'han afegit algunes funcionalitats i facilitat d'integració amb Eclipse.

https://github.com/rmtheis/tess-two

Aquesta llibreria s'haurà de compilar amb les NDK d'Android les quals es podem descarregar a:

http://developer.android.com/sdk/ndk/index.html

Per a compilar la llibreria s'hauran de seguir les següents instruccions:

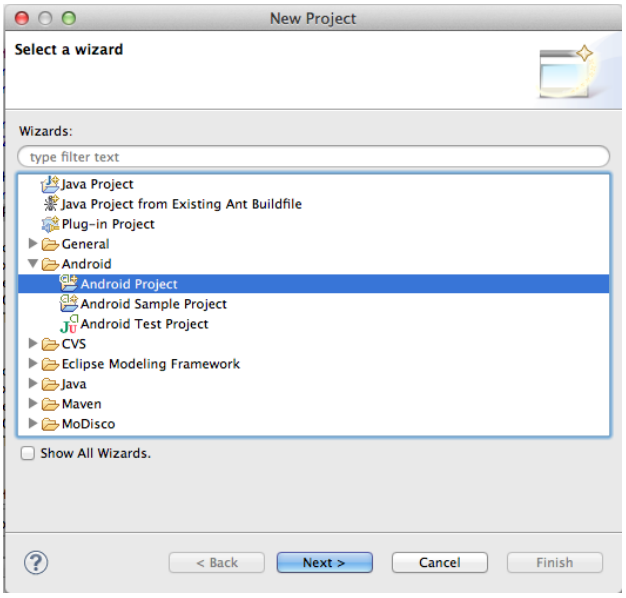
> *cd tess/tess-two*
> *ndk-build*
> *android update project --path .*
> *ant release*

Sinó es té el "ndk-build" al path s'haurà de cridar la ruta completa d'on s'ha descarregat, per a executar-lo.

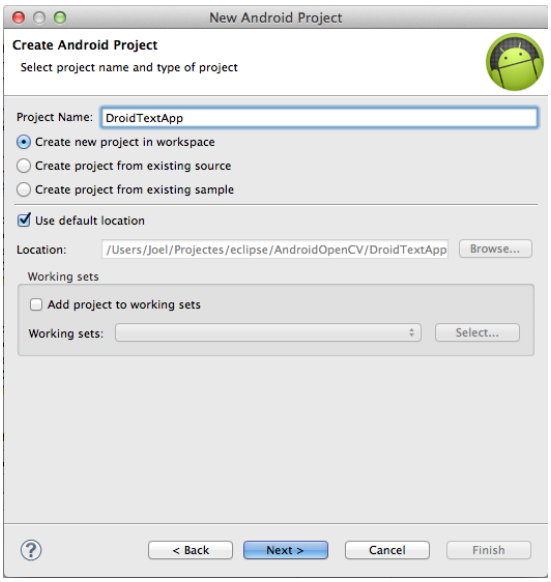
Ara es pot importar la llibreria de la mateixa forma que s'ha fet per a importar la llibreria d'OpenCV.

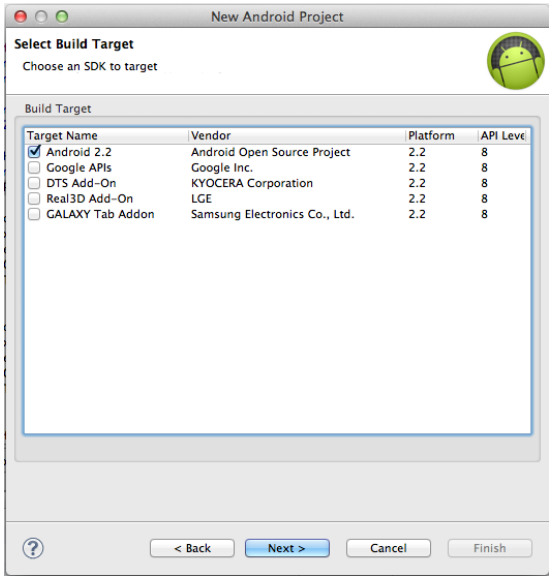
Un cop es tenen les dos llibreries correctament importades en l'espai de treball, es pot crear un nou projecte i afegir-hi les llibreries per a començar a treballar amb elles.

Es clica "File" --> "New Project" --> "Other" --> "Android" --> "Android Project"

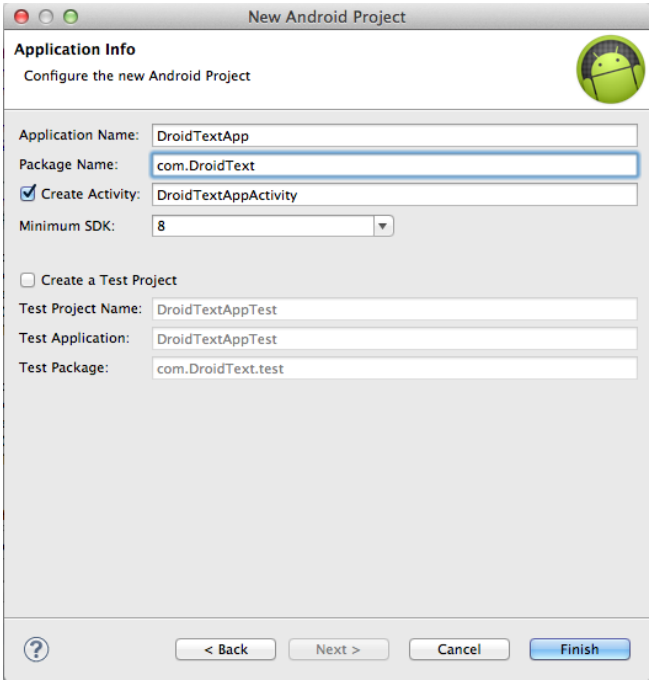


Se li dona un nom al projecte, i es selecciona el "target" desitjat, en aquest cas Android 2.2.

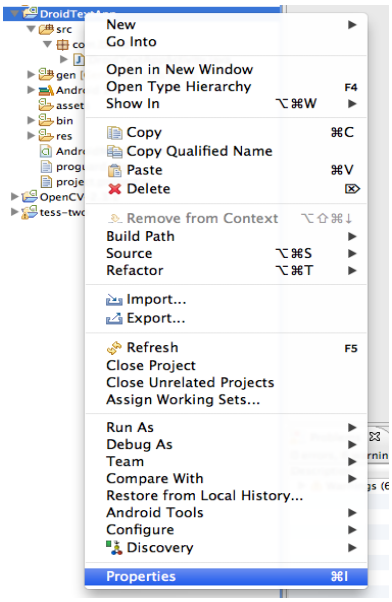




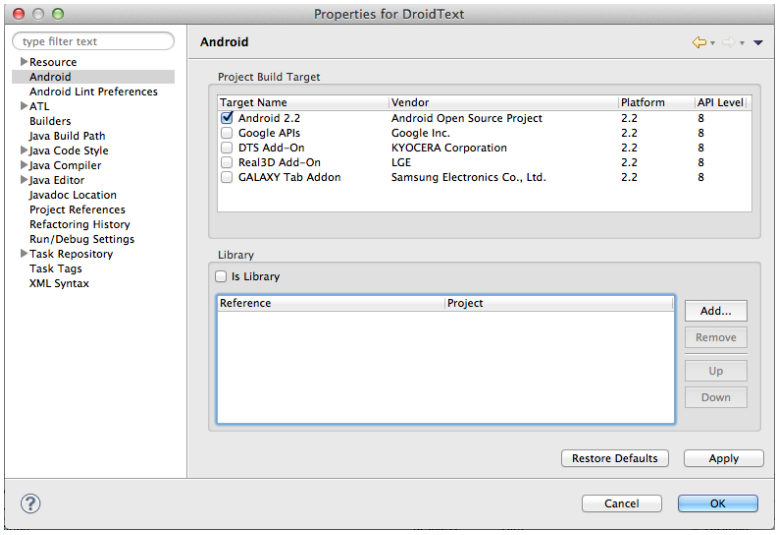
I per últim es crea l'activitat principal de l'aplicació:



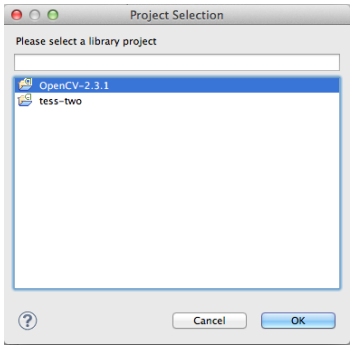
Ara s'han d'afegir les llibreries al nostre projecte. Es clica amb el botó dret sobre l'arrel del projecte (en el Package explorer) i es selecciona "Propietats".



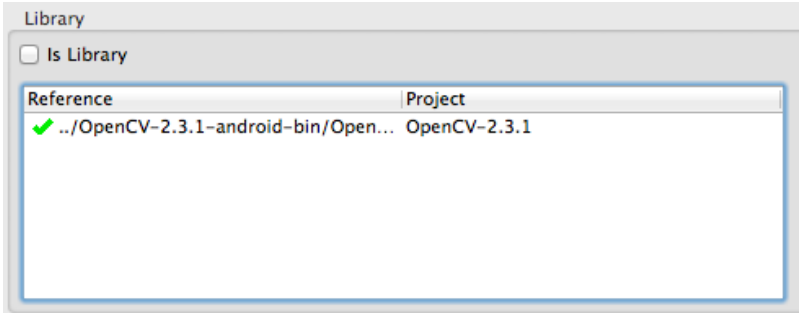
Es selecciona el menú "Android" i trobarem una vista de les llibreries que té el projecte i els seus "target".



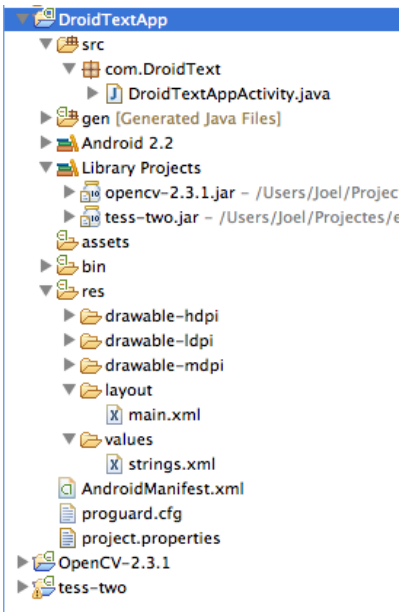
Es clica "Add ..." i s'afegeix la llibreria OpenCV.



Per comprovar que tot a funcionat correcte, hauria de sortir la llibreria en la taula corresponent. Repetir el procés per la llibreria del Tesseract.



Ara ja es té un projecte correctament configurat i es pot començar a utilitzar tots els avantatges de OpenCV i de Tesseract OCR per a dissenyar la nostra aplicació.



Si el que es vol es continuar des de el projecte aquí creat, en comptes de crear un projecte nou només s'ha d'importar el projecte de la mateixa manera en que s'han importat les llibreries.

10.2. Manual d'usuari DroidText

Que és DroidText?

DroidText és una aplicació de reconeixement de text en imatges per a dispositius mòbils amb sistema operatiu Android.

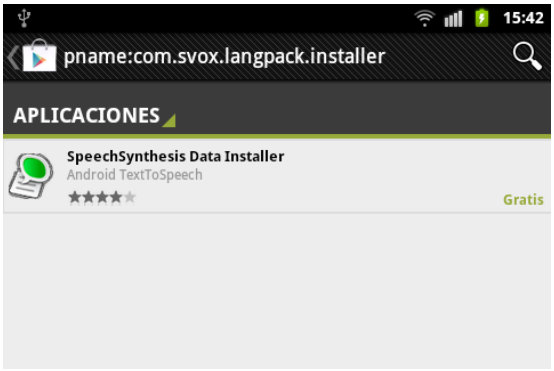
Que hem permet fer?

DroidText permet reconèixer qualsevol text en imatges en temps real, i permet les següents accions a dur a terme amb el text reconegut:

- Traduir-lo a diferents idiomes
- Escoltar-lo
- Enviar-lo per correu electrònic
- Compartir-lo en xarxes socials
- Compartir-lo per xarxes Bluetooth
- ...

Que cal configurar?

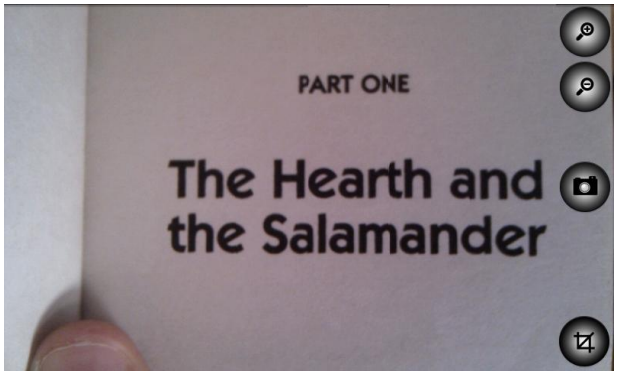
El primer cop que s'obre l'aplicació, aquesta crearà tots els directoris necessaris, i comprovarà que tinguem instal·lat un sintetitzador de veu, de no ser així, ens portarà a Google Play per a poder descarregar aquest sistema.



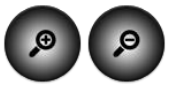
Un cop instal·lat el sintetitzador de veu, podrem canviar entre els diferents idiomes prement la tecla "menú" del nostre dispositiu i tot seguit el idioma escollit.

Com funciona?

DroidText mostra en tot moment una previsualització del que la càmera esta obtenint. Només ens hem de moure per l'entorn i enfocar l'escena a reconèixer.



Un cop en pantalla el motiu a capturar podem fer servir els següents botons:



Incrementa o disminueix el zoom de la càmera (si existeix)

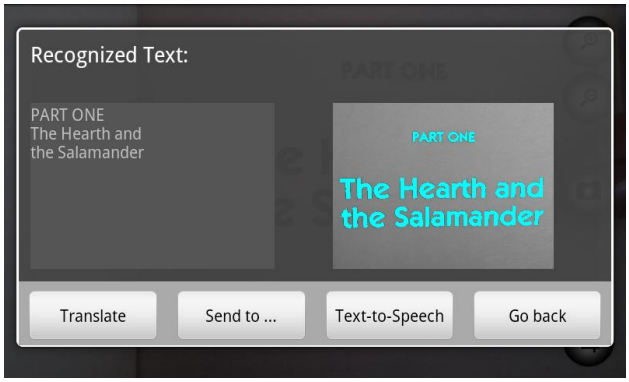
Comença el reconeixement de la escena en pantalla

Activa/Desactiva el retallat de la imatge en pantalla

El retallat de la imatge es mostra com un rectangle superposat sobre la imatge previsualitzada. Podem canviar la mida del rectangle arrossegant els seus vèrtex o costats.

Que són els resultats mostrats?

Un cop processada la imatge, una pantalla ens mostra el text reconegut així com la imatge amb el text reconegut pintat, per a compendre millor els resultats del procés.



Un cop tenim el text reconegut podem traduir-lo, enviar-lo o escoltar-lo, prement els botons corresponents.

10.3. Codi font

Es pot trobar el codi font en l'annex en format digital, separat en dos parts. El codi referent a la fase de desenvolupament, es troba en la carpeta amb aquest mateix nom, dins la subcarpeta *"Codi font"*. Referent al codi de l'aplicació *Android*, es pot trobar dins la carpeta *"Aplicació Android"*, en la subcarpeta *"Codi font"*.

10.4. Base de dades

Es poden trobar les imatges usades en les proves en l'annex en format digital, dins la carpeta *"Fase de desenvolupament"* en la subcarpeta *"Imatges"*. Aquesta està dividida en *"Original"*, que conté el set de proves original de ICDAR 2003, amb una simplificació dels directoris originals, la subcarpeta *"Selecció"* que conté el segon set de proves usat, i la subcarpeta *"Entrenament"*, que conté les imatges usades per l'entrenament de l'arbre de decisió.